SISTEMA DE COMUNICACIÓN INALÁMBRICO BASADO EN EL PROTOCOLO ZIGBEE

JESÚS RODRÍGUEZ LÓPEZ

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE ELECTRÓNICA

TRABAJO FIN DE GRADO PARA OPTAR AL TÍTULO DE GRADUADO EN INGENIERÍA DE SISTEMAS DE COMUNICACIONES

TUTORA DEL PROYECTO

CELIA LÓPEZ ONGIL

Madrid, 24-2-2015

Este proyecto va dedicado a mis padres Laura y Francisco que tanto me han apoyado a lo largo de mi vida haciendo posible que haya estudiado esta carrera, a mi novia Miriam por apoyarme cada día durante estos meses en los que he podido dedicar muy poco tiempo para ella, al resto de mi familia y a todas las personas que me aprecian con las que he compartido algún momento de mi vida.

Quiero agradecer a la Universidad Carlos III de Madrid como institución y a los profesores que he tenido a lo largo de mi carrera por la formación que me han brindado, especialmente a mi tutora del proyecto, Celia López Ongil, por haber dedicado parte de su tiempo en mí y a los técnicos de laboratorio César Vega Colado y Jesús Arroyo Beltrán.

ÍNDICE

LISTA DE FIGURAS

1. SUMMARY

In this project has been developed a communication system based on the ZigBee protocol, composed of a coordinator and two end devices. The coordinator node is the master device in the network, is responsible for creating it, manages the association of new end devices and is the center of the communication system which have a star topology. It is designed to monitor in a segment LCD screen both the temperature of each point where the end devices are placed and the network, showing a series of parameters that allows to check the status of the network and that everything works properly.

To develope the project has been used a platform of the Atmel company which incorporates the microcontroller ATmega256RFR2, also incorporates a wireless communication module which conforms to the IEEE 802.15.4, standard that reuses the ZigBee protocol for its physical and MAC layers. This microcontroller is compatible with the ZigBee stack provided by Atmel which has been used in this project for implementing the coordinator and end devices application program.

Has been designed and assembled a printed circuit board to connect the LCD screen with the coordinator and control it via software, a power stage has been implemented in a prototype board for each one of the end devices as well as the application software, services and drivers to have a strong and organized program architecture.

The target set for this project is to design and implement a communication network based on ZigBee protocol that serves as basis to further develop of an useful application which help to improve some aspect of life. The features that have been marked as a target for the communication system are:

1. Use the ZigBee communication protocol.

2. The network will have a star topology, in which the coordinator is the central node and all the end devices will be connected with him directly.

3. The network shall be of at least three nodes.

4. The end nodes will pair automatically with the coordinator, this means that the network must be managed autonomously, without manual configuration.

5. One of the nodes will be equipped with a LCD screen to display characteristic parameters of the network and to verify its proper operation. These parameters are: network size, link quality and packets sent by each device.

6. The nodes should not rely of energy supply from the mains, so it must be powered by a battery.

7. The energy consumption shall be the minimum possible, so the “sleep” function of the end devices must be implemented.

8. It must have a temperature sensor in the end devices. The sensed data will be transmitted to the coordinator periodically to monitor this environmental variable.

9. There should be an intuitive and simple human interface to control the information which is represented on the screen.

Zigbee protocol.

ZigBee communication protocol layers has a split architecture, the lowest layer is the physical, followed by the MAC layer, the network layer and finally the application layer. For the first two layers, physical and MAC, the protocol reuses the IEEE 802.15.4 standard so the application and network layers can be implemented differently by each manufacturer. With regard to the application layer, the ZigBee Alliance has standardized multiple profiles for applications such as remote control, input device, building automation, healthcare, home automation, saving energy and telecommunications services.

A ZigBee network offers a wide variety of design possibilities and can form star topologies, mesh and tree. In a star topology, the network is controlled by the central device which is called the coordinator node. The coordinator is responsible of the association process, syncronization and disassociation of devices in the network. All other devices, called end devices, are connected directly with the coordinator. In tree and mesh topologies, the coordinator is responsible for creating the network and select certain network parameters. In this case the network can be extended by using ZigBee routers. In tree topologies, routers transmits frames through the network using a hierarchical routing strategy and can implement beacon networks.

Functional description of the network.

It was decided to implement a non beacon ZigBee network with a star topology. The network is organized autonomously starting with the creation of the network by the coordinator. First, the network coordinator performs a scan of the 27 channels available for each of the 32 pages of channels. Through this process, the optimum channel is determined according to the estimation of the signal to noise ratio. Once the coordinator has chosen the channel to be used, waits for a beacon request from an end device to send a beacon frame.

This beacon send request will be transmitted by a device that wants to join the network in the scanning period to detect coordinators which are in the coverage range.

Once an end device has found a coordinator, it sends an association request which have to be confirmed by the coordinator assigning an free network address for the new device. End devices are most of the time in sleep mode, so when the association period has finished, the nodes adopt low power mode during an interval of 30 seconds, moment in which wake up and send the temperature data to the coordinator.

Data transmission is done directly, a device that wants to transmit a frame to the coordinator can do it once the channel is free and not have to wait to receive an indirect data request from the coordinator.

To provide a visual interface of the association and data frame transmission events, a LED in each device is used. In the coordinator, this LED produce a blink for a short time when a node is associated to the network and in the end devices the LED is on while they are awake and off when asleep, in this way is easy to know when data is transmitted.

Technology used.

To decide the technology to be used in this project has made a comparison between the technologies currently available on the market comparing the features and price of each and the feasibility of it to develop the project. It was decided to use the Atmel ATmega256RFR2 Xplained Pro evaluation kit. It is a platform that integrates the neccesary components to develop the project as connectors with I/O microcontroller pins, USB interface with integrated debugger on the board, a temperatura sensor, two mechanical buttons, two 16MH and 32KHz glasses and an antenna, reducing the development time. Also, is compatibile with the ZigBee software stack provided by Atmel.

The ATmega256RFR2 microcontroller that integrates the platform can be programmed by performing a Flash memory downliad through the USB interface connecting directly to a PC. It uses the integrated debugger which incorporates, with which it is possible to debug the program using the development enviroment Atmel Studio 6.2.

Hardware

The printed circuit board designed and implemented in this proyect has as main functions power the LCD pins with the necessary voltage levels using the input/output pins of the microcontroller and connect the LCD to the platform. To make the design of the board were performed analytical calculations necessary to determine the value of the components to be incorporated as well as laboratory tests to verify that the signals at the input of each LCD screen pin were generated correctly. The PCB designed has two layers, the top layer contains the LCD connector and the resistors that are connected to + Vlcd and the bottom layer

contains the resistors that are connected to GND. The printed circuit board is connected to the input/output microcontroller pins through three 90° angle connectors, located in the exacts positions so it can be assembled correctly.

The power stage has been implemented in a prototype board using the MC7805 and LM317 regulators and a battery as power source. The Atmega256RFR2 Xplained Pro platform must be supplied with 3.3V and 5V with a minimum and maximum current of 500mA and 2.5A respectively. The minimum supply current value corresponds to the minimum necessary to ensure the proper functioning of the platform, even connecting the maximum number of extension plates, and the maximum value is the current limit not to be exceeded to ensure platform safety. Given the current and voltage requirements, have been performed analytical calculations necessary to know the components value used in the circuit, also have been used the datasheets of both regulators.

Software

For the realization of this project have been developed drivers to control the segment LCD, several libraries for processing temperature data, user interface, short and long button press detection, temperature sensor services, TWI (2-Wires Interface) communication in addition to the coordinator and end devices application programs where communication is implemented using the ZigBee protocol.

The ZigBee communication has been developed using the Atmel Stack that implements the Physical and MAC layers providing all services required by the IEEE 802.15.4 standard.

The coordinator program is divided into two parts, the first controls the ZigBee communication and is performed in each iteration of an infinite loop located in the main function through mac_task () function. The other part is the application, which also runs on each iteration of this loop through the app_task () function to update the LCD. Also, the application includes all callback functions to communicate with the MAC layer. The processing temperature data is performed whenever such data are received from an end device, so that it runs only when needed. Furthermore, both beat detection and control of the user interface state machine are controlled using an interrupt executed every 20ms.

The end devices program is divided into two parts like the coordinator program, the first controls the ZigBee communication and is performed in each iteration of an infinite loop located in the main function through mac_task () function. The other part is the application, which also runs on each iteration of this loop through the app_task () function to establish communication with the coordinator sending the temperature data when the node is awake, communicate with the sensor to obtain the temperatura data and sleep the node. Also, the application program includes all callback functions to communicate with the MAC layer.

Results

The realized project meets the objectives marked at start, having managed to design and implement a ZigBee network of three nodes with the following characteristics:

- The network is properly set up by the coordinator, allowing the association and disassociation of end devices.
- The LCD screen is controlled by software providing a intuitive human interface controlled with the button on the platform.
- The Network size, temperature in the area where each end device is located, link quality between each coordinator-end device pair and number order of sent packets can be shown in the LCD screen.
- The Consumption of the end devices is 7.90mAh when the node is awake and 6.50mAh while asleep, and the coordinator has a consumption of 10.20mAh providing a battery life of 21h and 14h respectively.
- The network can cover 201m2 in an indoor area and 1963.5m2 in line of sight if the coordinator is placed in the center of the network.

2. INTRODUCTION

This project of the electronic technology department combines several of the subjects that have been studied during the career such as hardware, software and communication technologies. The work is study, design and implement all the necessary elements for a wireless communications network, capable of transmitting small amounts of data while consuming the least amount of energy possible.

The network consists of three nodes able to associate and communicate wirelessly, being formed by a coordinator and two end devices. The node assigned as coordinator is responsible for create and managing the network and has a human interface to visualize a series of parameters of the network in order to verify its correct operation.

The communication system developed in this project can serve as a basis for developing a wide range of applications in the future, such as industrial control systems without wiring and quick installation, monitoring systems of environmental variables with alarm functions for emergencies or any other application requiring low-capacity wireless communication.

The document is structured into four main parts, divided into 10 chapters. First, the problem to be solved is set and the study of the current technique is detailed in chapters 3 and 4. Second, selected wireless protocol is detailed as well as selected technology is justified in chapters 5 and 6. Subsequently, are explained in detail the hardware and software architectures designed and developed in Chapters 7 and 8. Finally, the results obtained are presented, as well as conclusions and future work lines in Chapters 9, 10 and 11.

3. MOTIVACIÓN Y OBJETIVOS

Las comunicaciones han revolucionado y están revolucionando el mundo, consiguiendo reducir la brecha digital y brindando mayores oportunidades a todas las personas, sin distinción de clases sociales gracias al mayor acceso a la información.

En este proyecto se plantea realizar una red de comunicaciones, que a diferencia de las grandes redes de comunicación de elevada capacidad, está enfocada a la transmisión de pequeños paquetes de datos; pero no por ello pasa a ser menos útil ya que tiene una serie de características que la hacen muy apropiada en varios entornos, como por ejemplo: su bajo coste debido a que la tecnología necesaria para su implementación no requiere una gran capacidad de procesamiento y su bajo consumo eliminando la necesidad de que la alimentación de los nodos sea suministrada por la red eléctrica y reduciendo el consumo energético.

Estas características hacen que una red de este tipo pueda ser desplegada con facilidad en cualquier lugar del mundo, como puede ser, por ejemplo, un bosque en el que se quieren monitorizar variables ambientales para la detección de incendios, o cualquier otra aplicación de utilidad para la humanidad que requiera una baja capacidad de transmisión de datos.

El objetivo marcado para este proyecto es realizar el diseño e implementar una red de comunicaciones basada en el protocolo ZigBee, que sirva como base para poder desarrollar posteriormente una aplicación que sirva de utilidad y ayude a mejorar algún aspecto de la vida. Las características que se han marcado como requisitos para el sistema de comunicaciones son las siguientes:

1. Será utilizado el protocolo de comunicaciones ZigBee.

2. La red tendrá una topología en estrella, donde el coordinador será el nodo central y todos los dispositivos finales estarán conectados con él directamente.

3. La red estará formada por al menos tres nodos.

4. Los nodos finales deberán asociarse automáticamente con el coordinador una vez encendidos, esto quiere decir que la red deberá gestionarse de forma autónoma, sin necesidad de configurarla manualmente.

5. Uno de los nodos estará equipado con una pantalla de tipo LCD, para poder visualizar parámetros característicos de la red en todo momento y poder verificar el correcto funcionamiento de la misma. Estos parámetros son: tamaño de la red, calidad de los enlaces y paquetes enviados por cada dispositivo.

6. Los nodos no deberán depender del suministro energético de la red eléctrica, por lo que deberán estar alimentados por algún tipo de batería.

7. El consumo energético deberá ser el mínimo posible, por lo que se deberá de utilizar la función de “sleep” (modo de bajo consumo) en los nodos cuando sea posible para disminuir el consumo de los mismos.

8. Se deberá disponer de un sensor de temperatura en los nodos finales. Los datos sensados serán transmitidos al coordinador de forma periódica para poder monitorizar esta variable ambiental.

9. Deberá existir una interfaz de usuario para poder controlar que información se quiere mostrar en cada momento en la pantalla. Ésta deberá ser sencilla e intuitiva.

4. ESTUDIO DEL ESTADO ACTUAL DEL PROTOCOLO ZIGBEE

Las redes inalámbricas se encuentran en un período de crecimiento frente a las redes cableadas, sobre todo en el sector de la automatización industrial y de edificios. Hay varios factores que favorecen este hecho como la reducción de costes en cableado y mano de obra, el bajo coste de la tecnología (en el caso de ZigBee), la reducción del consumo energético (en el caso de ZigBee) y la sencillez de implementación de este tipo de redes sin necesidad de obtener permisos de obra en la zona donde se quiere desplegar.

Todas estas características hacen muy atractivas las soluciones de red inalámbricas frente a las cableadas colocando al protocolo ZigBee en una posición ventajosa a día de hoy. Es tal lo que se espera de esta tecnología que numerosos fabricantes decidieron apostar por ella creando así la llamada ZigBee Alliance, formada por promotores como: Comcast, Freescale™, Itron, Kroger, Landis Gir, Legrand®, NXP, Philips, Schneider, Silicon Laboratories y Texas Instruments a la que cada vez se unen más empresas como participantes o simplemente como miembros. El protocolo Zigbee se apoya en un estándar internacional (IEEE 802.15.4) en las primeras capas del modelo OSI, capas física y de enlace. Así mismo, el protocolo define la capa de red y de aplicación. Todos los fabricantes deben cumplir el estándar, pero las capas más altas del modelo OSI no tiene por qué ser compatibles entre todos los productos Zigbee de distintos fabricantes. La alianza Zigbee ha establecido un conjunto de definiciones y criterios que permite utilizar dispositivos Zigbee de distintos fabricantes sin problemas de incompatibilidad.

El protocolo Zigbee puede operar en diferentes frecuencias (868/915MHz y 2,4GHz), siendo la de 2,4GHz la más extendida, conocida como la banda ISM.

Pero es cierto que en algunos aspectos el estándar tiene ciertas limitaciones con respecto a algunos de sus competidores que obstaculizan su entrada y consolidación en el mercado. Sus principales competidores son Bluetooth, Wi-Fi, Z-Wave, KNX, Wireless HART y 6LoWPAN. Hay que tener en cuenta que sus competidores también operan en la banda ISM. La Figura XX muestra una comparativa entre las diferentes características de los estándares inalámbricos.

	Wi-Fi	**ZigBee**	**Z-wave**	**KNX**	**6LoWPAN**	**Wireless HART**
Topología	Estrella	Estrella, malla, árbol	Malla	Estrella, malla, árbol	Estrella, malla	Estrella, malla
Consumo	Muy elevado	Muy bajo	Muy bajo	Muy bajo	Muy bajo	Muy bajo
Tasa de transmisión	2Mbps-1.3Gbps	250Kbps	40Kbps	1.2Kbps	250Kbps	250Kbps

Rango	20m	10-300m	30m	600m	800m	200m

Tabla 1. Comparativa entre las distintas tecnologías inalámbricas de corto alcance.

La limitada capacidad de transmisión que soportan los enlaces ZigBee, hasta 250 kbps[1], aleja al protocolo de aplicaciones relacionadas con redes de acceso o de transmisión de datos multimedia, siendo este mercado acaparado por los estándares Wi-Fi y Bluetooth. Pero, como es lógico, la proliferación de protocolos hace que cada uno sea más adecuado para unas aplicaciones concretas, por lo que ZigBee se ajusta más a las redes de sensores orientadas a domótica e inmótica.

El protocolo tiene varios perfiles, cada uno orientado a una aplicación diferente para facilitar el desarrollo de las soluciones con ZigBee. Todos los perfiles del protocolo están basados en la misma capa física y MAC, modificando la capa de red y la capa de aplicación del mismo para así particularizar cada uno de los diferentes perfiles. Uno de ellos es ZigBee RFACE, que está orientado al control remoto y proporciona una serie de comandos habituales en este tipo de sistemas para poder controlar los diferentes dispositivos de una vivienda. Este perfil del protocolo no contempla la funcionalidad de enrutado de paquetes, ya que simplemente existen dispositivos controladores que deben establecer comunicación de forma directa con los dispositivos controlados. Otro de los perfiles es el ZigBee PRO, que soporta enrutamiento de paquetes con el que se pueden implementar una amplia variedad de aplicaciones como indica la posterior imagen. Por último se encuentra el perfil ZigBee IP, que permite incorporar en la red un dispositivo de pasarela con una red IP para poder transmitir directamente los datos a través de internet mediante un *router* IP. Es probable que este último perfil tenga un papel importante en el futuro "internet de las cosas". En la figura 1 se puede ver una clasificación de los perfiles del estándar ZigBee y sus aplicaciones.

[1] Kilobits por segundo, es una medida para indicar la velocidad de transimisión de datos digitales

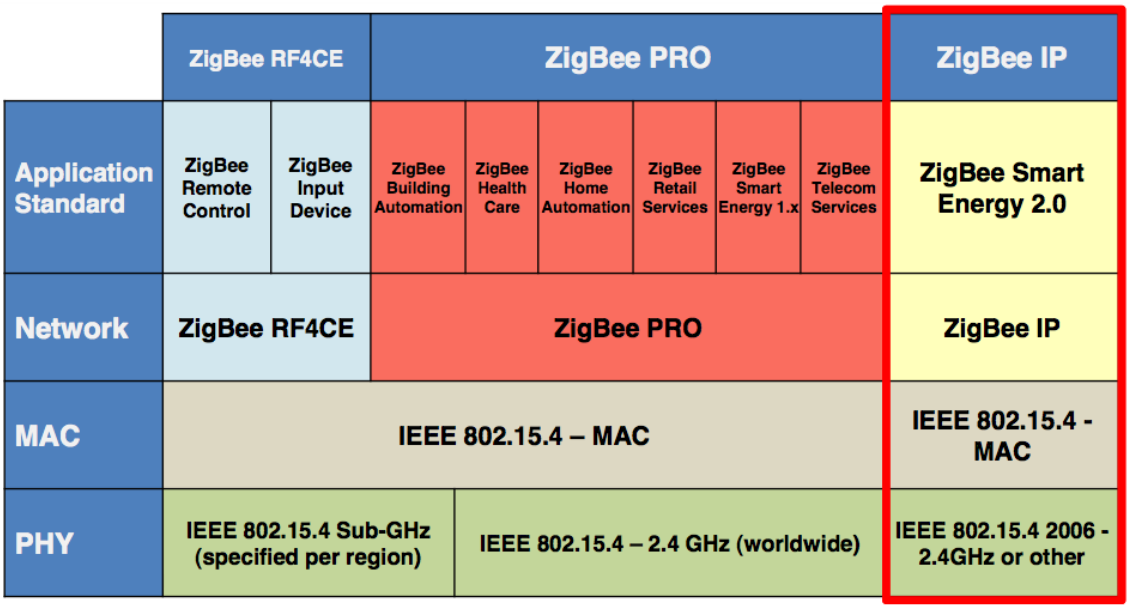

Figura 1. Aplicaciones de los perfiles del protocolo ZigBee.

El desarrollo del protocolo y los diferentes perfiles hacen que ZigBee sea cada vez más utilizado por consumidores frente a su utilización en entornos comerciales e industriales, relación que se espera sigua aumentando en el futuro. En la figura 2 se puede comprobar que la relación entre la utilización del protocolo en entornos comerciales e industriales y consumidores es cada vez más dominante en el sector de consumo.

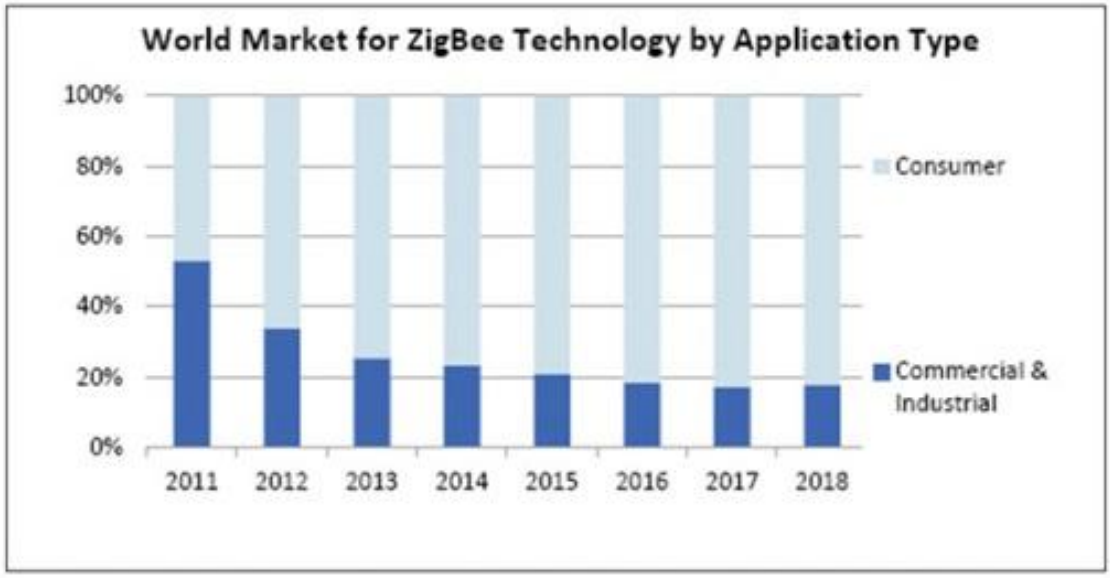

Figura 2. Relación de utilización del protocolo ZigBee entre usos comerciales e industriales y de consumidor.

4.1. TECNOLOGÍAS DISPONIBLES ZIGBEE®

Actualmente podemos encontrar módulos Zigbee® de la mano de fabricantes como Digi International, Telegesis, Atmel o NXP que otorgan ciertas facilidades en el desarrollo de aplicaciones, proporcionando una versión del *Stack Zigbee* certificado, aunque no todos los fabricantes ofrecen la base de software de forma gratuita. Generalmente estos módulos tienen un coste relativamente bajo si lo comparamos con otras tecnologías inalámbricas dependiendo del nivel de desarrollo del mismo.

Un módulo Zigbee® se compone de cuatro partes fundamentales: el sistema de comunicación inalámbrica (radiofrecuencia), el gestor de las comunicaciones (implementación del protocolo), el procesado de la información recibida o generada (sensado, actuadores, etc) y el sistema de alimentación. La manera más básica de implementar un módulo Zigbee® es mediante un microcontrolador comercial. Esta es es la opción que más tiempo requiere teniendo que desarrollar casi toda la parte hardware, incluyendo la parte de radio-frecuencia que suele ser la más problemática. Por otra parte, una manera más eficaz, bastante más rápida, pero más cara de implementar un módulo Zibgee®, es utilizar uno de los módulos existentes en el mercado con el hardware ya desarrollado , que facilita la programación/depuración del microcontrolador, incluyendo conectores e integrados de utilidad como pueden ser sensores, una memoria EEPROM, un depurador embebido aparte de diodos LED y botones de programación software o inicialización.

A diferencia de las opciones anteriores, los módulos más comunes incorporan solamente el microcontrolador junto con un bloque de radio-frecuencia formado por líneas de transmisión y una o varias antenas. A continuación se realizará una descripción de los módulos ZigBee que actualmente están en el mercado y que son utilizados por las empresas para desarrollar sus aplicaciones indicando sus principales características.

Describe lo que es el Stack y dí que a partir de ahora lo vas a llamar así.

4.1.1. XBee DE DIGI INTERNATIONAL

Figura 3. Módulo ZigBee XBee.

Hay diferentes modelos de este módulo de transmisión por RF fabricado por la empresa *Digi International*. Cada uno de ellos tiene diferentes características en cuanto a la topología, el rango de transmisión o los MCU que incluyen. De todos ellos el que ofrece mejores prestaciones es el modelo “Programable XBee-PRO ZB” que incluye un microcontrolador con arquitectura HCS08 con 32 KB Flash / 2 KB RAM de memoria dedicadas al programa de aplicación del usuario y que permite la creación de redes ZigBee en retícula o malla.

Sus principales características son:

- Rango en interiores: 90 m
- Rango en exterior (LOS): 1500 m
- Potencia transmitida: 10 mW (+10 dBm)
- Sensibilidad: -102 dBm
- Interfaces I/O: 3.3V CMOS UART, SPI, I2C, PWM, DIO, ADC
- Alimentación: 2.7 - 3.6VDC

- Corriente de transmisión: 220 mA
- Corriente de recepción: 62 mA
- Corriente en modo de bajo consumo: 4uA

Estos dispositivos están configurados y programados de fábrica para establecer comunicación sin necesidad de que el usuario haga prácticamente nada, es decir, estos dispositivos se encargarían en un nodo de una red ZigBee de la comunicación, por lo que debe conectarse por puerto serie con otro microcontrolador que se encargue del procesamiento de los datos sensados o enviados.

Es posible conectar una red XBee una red más amplia tipo Ethernet, por medio de los Gateways que proporciona Digi Internacional llamados XBee Gateway. Estos dispositivos proporcionan una interfaz entre la red ZigBee y una red IP a un número ilimitado de nodos XBee, así como permiten configurar y gestionar una red XBee. Estos dispositivos están programados en el lenguaje de programación *Python*, existiendo un entorno de desarrollo de código abierto para facilitar el desarrollo de la aplicación.

4.1.2. ETRX357 DE TELEGESIS

Figura 4. Módulo ZigBee ETRX3 de Telegesis.

Las series ETRX3 de Telegesis son módulos ZigBee a 2.4 GHz de baja potencia que integran un transceptor compatible con IEEE 802.15.4 que dispone de un máximo de 192KB de memoria Flash, 12KB de RAM y varios periféricos avanzados.

El ETRX3 utiliza una arquitectura eficiente que supera los requisitos de rango dinámico impuestos por el estándar IEEE 802.15.4 en más de 15 dB. El filtrado de canal integrado

permite una robusta coexistencia con otros estándares de comunicación en el espectro de 2,4 GHz, como IEEE 802.11 y Bluetooth.

Estos dispositivos de Telegesis incluyen un microprocesador ARM Cortex M3 que está optimizado para un alto rendimiento, bajo consumo de energía, y la utilización eficiente de la memoria, que lo hace ideal para su uso en aplicaciones de ZigBee.

Para mantener los requisitos de tiempo estrictos impuestos por el estándar 802.15.4 el ETRX3 integra una serie de funciones de control de acceso al medio (MAC) en el manejo de la transmisión y recepción automática de ACK, retardo de reenvío automático, evaluación del canal para la transmisión, así como filtrado automático de los paquetes recibidos.

Este módulo trabaja desde una alimentación de 2.1V hasta 3.6V. En el modo de sueño profundo, el consumo se reduce a 800nA y además se reduce a 400nA si la función de auto activación no está habilitada.

Telegesis ofrece todo el software ya presente en otros módulos de ZigBee así como el conjunto de comandos que permite al diseñador utilizar la funcionalidad de la pila EmberZNet sin necesidad de implementar código embebido complejo. Para programar y depurar el microcontrolador que incorpora este módulo es necesario conectar una interfaz de depuración utilizando la interfaz JTAG del microcontrolador.

4.1.3. ATZB-S1-256-3-0-C DE ATMEL

Figura 5. Módulo ZigBee ATZB-S1-256-3-0-C de Atmel.

ATZB-S1-256-3-0-C es un módulo ZigBee compacto y de bajo consumo de la empresa Atmel. Basado en la plataforma de hardware de señal mixta de Atmel, este módulo usa el

microcontrolador ATmega256RFR2 con arquitectura AVR de 8 bits y un transceptor para la banda ISM de 2,4 GHz. El *transceptor* de radio proporciona velocidades de datos de 250 Kb/s hasta 2 Mb/s, además de una alta sensibilidad en recepción y una potencia de transmisión de 3.6dBm.

El módulo tiene un conector MS-147 RF que puede ser utilizado como un puerto de prueba de RF. La antena de chip integrada está adaptada al módulo por lo que facilita el desarrollo de las aplicaciones suponiendo una reducción del tiempo empleado en el diseño.

Es compatible con la pila IEEE 802.15.4/ZigBee que soporta auto-reparación y auto-organización de una red mallada, al tiempo que optimiza el tráfico de red y reduce el consumo de energía.

Las características del módulo ATZB-S1-256-3-0-C de Atmel, de las cuales hay que resaltar la elevada memoria de la que dispone el MCU: 256 K Bytes de Flash, 8 K Bytes de EEPROM y 32 K Bytes de SRAM lo que permitiría implementar el software de la aplicación en el mismo microcontrolador sin necesidad de añadir otra unidad de procesamiento para este fin y la capacidad de soportar redes malladas, son las siguientes:

- Tamaño: 30.0 × 20.0mm
- Sensibilidad: -97dBm
- Potencia de transmisión: +3.6dBm
- Consumo de energía:
  - 9.6mA en modo RX
  - 16.4mA en modo TX
  - 0.6µA en modo de reposo.
- Microcontrolador AVR® de 8-bit
- Memoria:
  - 256K Bytes flash
  - 8K Bytes EEPROM
  - 32K Bytes SRAM
- Interfaces: 4 SPI, 4 TWI, ISP, JTAG, 2 comparadores analógicos, UART, USART, 31 pines configurables como GPIO.
- Periféricos: Temporizador, PWM, 4 módulos ADC

- Entrada de reloj externo
- Salida de reloj interno
- Capacidad para utilizar la dirección MAC en la EEPROM interna
- *Transceptor* compatible con IEEE® 802.15.4
- Banda ISM de 2,4 GHz
- Punto de prueba de RF utilizando el conector RF MS-147
- Capacidad de red en malla

4.1.4 JN5139-001-MXX de NXP

Figura 6. Módulos ZigBee JN5139-001-MXX de NXP.

Este módulo de la compañía NXP es uno de los más interesantes debido a los distintos modelos que podemos encontrar según los requerimientos de la aplicación y a las prestaciones del microcontrolador que incorpora.

Son varios los modelos NXP ha fabricado, diferenciándose principalmente en la parte de radio-frecuencia. El modelo más básico es el JN5139-xxx-M00. Dispone de una antena integrada con la que se consigue un rango de transmisión de 1km, la misma distancia que podemos conseguir con los modelos JN5139-xxx-M01/M03 que incorporan un conector SMA pero no tienen amplificador. Los modelos más avanzados son los JN5139- xxx-M02/M04 que incorporan un conector SMA junto con un amplificador de potencia y un amplificador de bajo ruido para incrementar el rango de transmisión hasta los 4Km.

Las características del módulo en función del modelo y las características del microcontrolador son las siguientes:

Módulo:

- Compatible con el estándar IEEE802.15.4 y el protocolo ZigBee
- Alimentación de 2.7-3.6V
- Corriente en sleep (con temporizador de sleep activo) 2.6µA
- JN5139-xxx-M00/01/03
    - Hasta 1km de rango
    - M00: antena integrada
    - M01: conector SMA
    - M03: conector uFl
    - Sensibilidad del receptor: -96dBm
    - Potencia de TX: +2.5dBm
    - Corriente de TX : 37mA
    - Corriente de RX: 37mA
    - Dimensiones: 18x30mm
- JN5139-xxx-M02/04
    - Hasta 4Km de rango
    - M02: conector SMA
    - M04: conector uFl
    - Sensibilidad del receptor: -100dBm
    - Amplificador de potencia
    - LNA
    - Potencia de TX: +19dBm
    - Corriente de TX: 125mA
    - Corriente de RX: 45mA

- Dimensiones: 18x41mm

Microcontrolador:

- CPU de 16MHz 32-bit
- 96KB RAM, 192KB ROM
- Periféricos : 4-entradas ADC de 12-bit, 2 DAC 11-bit, 2 comparadores, sensor de temperatura, 2 Temporizadores/Contadores de aplicación, 3 Temporizadores del sistema
- Interfaces: 2 UARTs (una para depuración), puerto SPI, TWI, 21 pines GPIO

Las características más destacables de este módulo son su elevada potencia de transmisión, lo que permitiría implementar aplicaciones en las que sea necesario cubrir una zona extensa reduciendo los costes. Además, el enlace tendrá mejor calidad si los nodos son colocados a una distancia intermedia relativa al máximo alcance, siendo esta una distancia muy considerable para un enlace ZigBee y la arquitectura de 32 bits del microcontrolador junto con la capacidad de memoria. Los 192 KB de memoria flash permite la integración de los *stacks* de ZigBee tanto para una topología punto a punto como mallada y los 96 K Bytes de RAM permiten integrar funcionalidades de *router* y de control, ya que estas aplicaciones necesitan utilizar una gran cantidad de variables que deben ser almacenadas en RAM.

5. PROTOCOLO ZIGBEE

5.1. DESCRIPCIÓN GENERAL

ZigBee es un protocolo de comunicación de bajo coste y bajo consumo desarrollado por la Alianza ZigBee. Podemos encontrar soluciones que lo adoptan en electrónica de consumo, automatización de hogares y edificios, control industrial, periféricos de PC, aplicaciones médicas, juguetes y juegos.

La arquitectura del protocolo ZigBee está basada en una serie de capas. Cada capa implementa un conjunto de servicios específicos para la capa superior. Una entidad de datos proporciona servicios relacionados con la transmisión de datos y una entidad de gestión proporciona todos los demás servicios. Cada entidad de servicios incorpora una interfaz con la capa superior que se denomina Service Access Point (SAP), a través del cual la capa superior podrá hacer uso de unos servicios fundamentales para conseguir las especificaciones del estándar en el caso de las capas física y MAC.

Las dos primeras capas están definidas por el estándar IEEE 802.15.4, la física (PHY) y la de control de acceso al medio (MAC). La alianza ZigBee definió sobre esta base una capa de red y la interfaz para la capa de aplicación. La interfaz de la capa de aplicación consiste en la sub-capa de soporte de la aplicación (APS) y los objetos de dispositivo ZigBee (ZDO) por lo que la aplicación desarrollada podrá utilizar los servicios proporcionados por esta interfaz.

El protocolo IEEE 802.15.4 puede operar en dos rangos de frecuencia distintos: 868-915 MHz y 2.4GHz. La más utilizada es la banda de mayor frecuencia que se utiliza mundialmente, mientras que la banda de 900MHz se utiliza mayormente en Estados Unidos y Australia.

La capa MAC del estándar IEEE 802.15.4 controla el acceso al radio canal mediante el mecanismo CSMA-CA. Este mecanismo consiste en escuchar el canal de transmisión para medir la energía y poder detectar si algún otro dispositivo está transmitiendo en ese mismo momento para evitar así colisiones entre transmisiones de distintos dispositivos. Entre sus funciones se encuentran la transmisión de tramas baliza, la sincronización y el deber de proporcionar un mecanismo de transmisión que garantice una comunicación eficiente, es decir, que los dispositivos compartan el medio de transmisión de forma que no se produzcan retardos excesivos o tasas de errores elevadas. La figura 7 muestra la arquitectura del protocolo ZigBee.

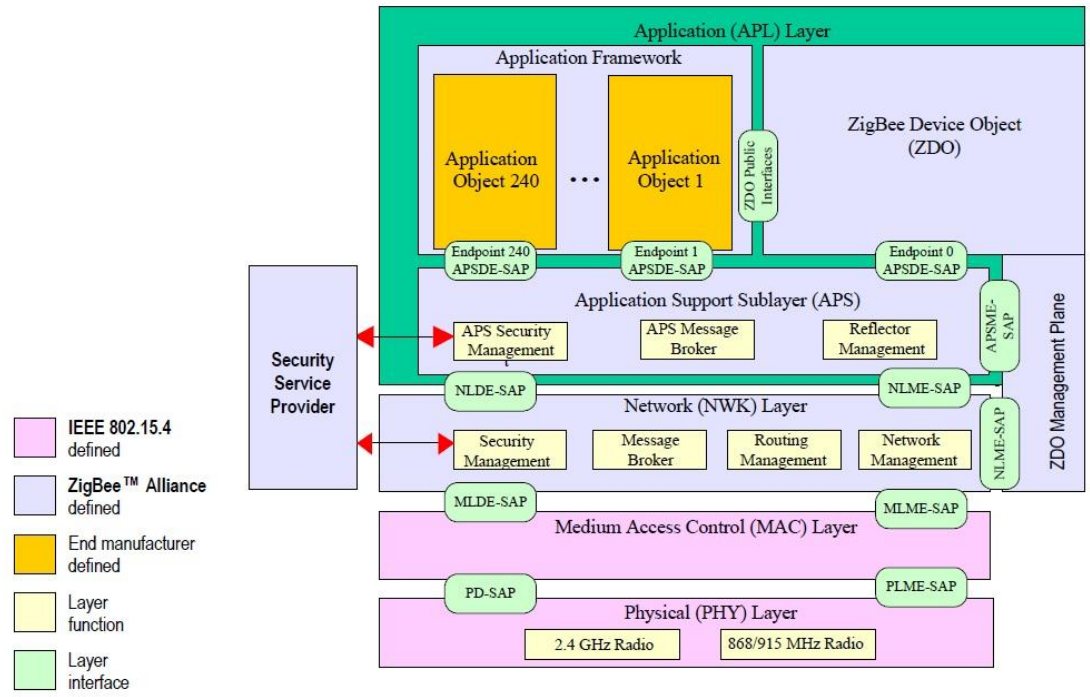

Figura 7. Esquema de la arquitectura de la pila ZigBee. Imagen obtenida de "Atmel AVR2025: IEEE 802.15.4 MAC Software Package - User Guide"

La figura 7 muestra cómo está organizado el protocolo desde la capa física hasta la capa de aplicación. Este esquema ofrece una visión clara de la estructura que debe tener cualquier Stack de *software* del protocolo ZigBee. La idea es sencilla, cada capa está formada por una serie de funciones, máquinas de estado, etc. cuyo objetivo es satisfacer los requisitos y especificaciones de dicha capa.

En el nivel más bajo de la arquitectura se encuentra la capa física, formada por el hardware que generalmente será un microcontrolador y un transmisor de radio-frecuencia y los procedimientos y funciones que se ejecutan en el microcontrolador para gestionar la comunicación en dicho transmisor (*drivers)* que lo controlan mediante los registros del MCU y ofrecen una serie de servicios a la capa MAC a través de las interfaces *Physical Data Service Access Point* (PD-SAP) y *Physical Data Service Access Point* (PLME-SAP). Estas interfaces denominadas *Single Access Point (*SAP), básicamente son la *Application Procedural Interface* (API) que se podrá utilizar desde la capa inmediatamente superior para utilizar los servicios implementados en la capa. El estándar distingue dos tipos de interfaces o SAP, la que ofrece servicios de transmisión de datos y la que ofrece servicios de gestión.

Las interfaces relacionadas con la parte de transmisión de datos son:

- PD-SAP: Physical Data – Service Access Point
- MLDE-SAP: MAC Layer Data Entity – Service Access Point

- NLDE-SAP: Network Layer Data Entity – Service Access Point

Por otro lado, las que están relacionadas con servicios de gestión son:

- PLME-SAP: Physical Data – Service Access Point
- MLME-SAP: MAC Layer Data Entity – Service Access Point
- NLDE-SAP: Network Layer Data Entity – Service Access Point

La capa de red de ZigBee permite realizar topologías en estrella, árbol y malla. En una topología de estrella, la red es controlada por el dispositivo central al que se le denomina nodo coordinador. El coordinador se encarga de incorporar, mantener y eliminar dispositivos en la red. Todos los demás dispositivos, llamados dispositivos finales, se conectan directamente con el coordinador.

En topologías en árbol y en malla el coordinador es el responsable de crear la red y de seleccionar ciertos parámetros de la red. En este caso la red puede ser extendida mediante el uso de *routers* ZigBee. En redes con topología en árbol, los *routers* transmiten tramas a través de la red utilizando una estrategia de enrutado jerárquica, pudiendo implementar redes balizadasL Las redes balizadas consisten en el establecimiento de unos períodos de tiempo de transmisión y de reposo. La transmisión de una serie de tramas por parte del coordinador sirven de guía para el resto de dispositivos para permanecer sincronizados respetando los intervalos de transmisión y de reposo que marca el coordinador.

5.2. CAPA FÍSICA

La capa física del protocolo Zigbee® está definida en el estándar IEEE 802.15.4 y es responsable de las siguientes tareas:

1. Activación y desactivación del transceptor de radio.
2. Detección de Energía del canal actual.
3. Indicador de calidad de enlace para paquetes recibidos.
4. Evaluación de borrado de canal para la detección de portadora de acceso múltiple con el mecanismo de prevención de colisiones CSMA-CA.
5. Selección de la frecuencia de canal.

6. Transmisión y recepción de datos.

5.2.1. MODULACIONES Y TASAS DE TRANSMISIÓN DE DATOS

La norma específica la modulación que debe utilizarse para cada una de las tres bandas de frecuencia posibles en el protocolo Zigbee®:

1. Una modulación BPSK (*binary phase-shift keying*) de espectro ensanchado por secuencia directa en el espectro de 868/915 MHz.

2. Una modulación O-QPSK (*offset quadrature phase-shift keying*) de espectro ensanchado por secuencia directa en el espectro de 868/915 MHz.

3. Modulaciones ASK (*amplitude shift keying*) y BPSK de espectro ensanchado por secuencia paralela en el espectro de 868/915 MHz.

4. Una modulación O-QPSK de espectro ensanchado por secuencia directa.

Además de la modulación BPSK en las bandas de 868/915 MHz, se especifican dos modulaciones de mayor velocidad de datos opcionales para estas bandas z, ofreciendo una solución de compromiso entre la complejidad y velocidad de datos. Las modulaciones opcionales ofrecen una velocidad de datos mucho mayor que la BPSK, que prevé 20 Kb/s en la banda de 868 MHz y 40 Kb/s en el 915 MHz banda. La modulación ASK ofrece velocidades de datos de 250 Kb/s en ambas bandas, que es igual a la de la banda de 2,4 GHz. La modulación O-QPSK, que ofrece un esquema de señalización idéntica a la de la banda de 2,4 GHz, ofrece la misma velocidad de datos en las bandas de 915 MHz y 2,4 GHz y una velocidad de datos de 100 kbps en la banda de 868 MHz.

La tabla 2 muestra las bandas de frecuencia junto con sus respectivas tasas de transmisión de datos.

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
868/915 (optional)	868–868.6	400	ASK	250	12.5	20-bit PSSS
	902–928	1600	ASK	250	50	5-bit PSSS
868/915 (optional)	868–868.6	400	O-QPSK	100	25	16-ary Orthogonal
	902–928	1000	O-QPSK	250	62.5	16-ary Orthogonal
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Tabla 2. Bandas de frecuencia y tasas de transmisión de datos. Imagen obtenida de "IEEE Std 802.15.4k™-2013"

5.2.2. ASIGNACIÓN DE CANALES

El espectro utilizado por el estándar está dividido en 32 páginas de canales y cada una de ellas está formada por un conjunto de 27 canales. Las 32 páginas de canales quedan especificadas por los 5 bits más significativos de un valor entero de 32 bits y los 27 bits restantes son usados como máscara para especificar cada uno de los 27 canales dentro de una página.

El conjunto de canales está dividido en las tres bandas de frecuencia y para cada una de ellas hay un diferente número de canales con un anchura espectral distinta. Dieciséis canales están disponibles en la banda de 2.4 GHz con una anchura espectral por canal de 5MHz, 10 en la banda de 915 MHz con una anchura de 2 MHz y un canal en la banda de 868 MHz. Las frecuencias centrales de los canales están definidas de la siguiente forma:

- Fc = 868.3 MHz , para k = 0
- Fc = 906 + 2 (k – 1) MHz , para k = 1, 2, …, 10
- Fc = 2405 + 5 (k – 11) MHz, para k = 11, 12, …, 26

Las páginas de canales 3-31 están reservadas para uso futuro, al igual que las bandas de 2.4 GHz en las páginas 1 y 2. En la tabla 3 se encuentran las páginas de canales y los números de canal junto con una descripción.

Channel page (decimal)	Channel page (binary) (b_{31}, b_{30}, b_{29}, b_{28}, b_{27})	Channel number(s) (decimal)	Channel number description
0	0 0 0 0 0	0	Channel 0 is in 868 MHz band using BPSK
		1–10	Channels 1 to 10 are in 915 MHz band using BPSK
		11–26	Channels 11 to 26 are in 2.4 GHz band using O-QPSK
1	0 0 0 0 1	0	Channel 0 is in 868 MHz band using ASK
		1–10	Channels 1 to 10 are in 915 MHz band using ASK
		11–26	Reserved
2	0 0 0 1 0	0	Channel 0 is in 868 MHz band using O-QPSK
		1–10	Channels 1 to 10 are in 915 MHz band using O-QPSK
		11–26	Reserved
3–31	0 0 0 1 1 - 1 1 1 1	reserved	Reserved

Tabla 3. Páginas de canales y número de canales. Imagen obtenida de "IEEE Std 802.15.4k™-2013"

5.2.3. SERVICIOS DE LA CAPA FÍSICA

La capa física ofrece un servicio de datos a través del PD-SAP y un servicio de gestión a través del PLME-SAP. El servicio de datos posibilita el transporte de MPDUs entre las entidades de la capa MAC. Esta interfaz está formada por una serie de primitivas o funciones fundamentales que pueden ser utilizadas desde la capa superior y que proporcionan una serie de servicios básicos.

Las primitivas que incorpora el PD-SAP son:

- **PD-DATA.request:** Solicita la transmisión de un MPDU desde la capa MAC a la entidad de datos de la capa física.

- **PD-DATA.confirm:** Notifica el resultado de la solicitud de transmisión de un MPDU. Los diferentes resultados que pueden darse son: SUCCESS, RX_ON, TRX_OFF, o BUSY_TX.

- **PD-DATA.indication:** Notiffca a la capa MAC que se ha recibido un MPDU, es decir, que se ha recibido una trama de otro dispositivo.

El punto de acceso al servicio de gestión PLME-SAP permite la transferencia de comandos entre la entidad de gestión de la capa física PLME y la entidad de gestión de la capa MAC MLME. Las funciones primitivas que forman este "*Service Access Point*" son:

- **PLME-CCA.request:** Solicita una valoración del estado del canal. Este servicio es necesario para la implementación del mecanismo CSMA-CA en la capa MAC.

- **PLME-CCA.confirm:** Esta función de *callback* es ejecutada por la capa física tras realizar una petición de *Clear Channel Assessment* (CCA) e indica si el canal está ocupado o listo para transmitir.

- **PLME-ED.request:** Solicita que se realice una estimación del nivel de energía de un canal.

- **PLME-ED.confirm:** Esta función de *callback* es ejecutada por la capa física tras realizar una solicitud de medida del nivel de energía del canal. En sus parámetros contiene el resultado de la solicitud junto con el nivel de calidad estimado del canal.

- **PLME-GET.request:** Solicita información sobre un atributo de la capa física, esta función se conoce con un "*Getter*" en términos de programación. Puede solicitar el

valor de varios atributos como: el canal actual en el que se encuentra, los canales soportados, el nivel de potencia de transmisión, la página de canales actual, la duración máxima de trama y los símbolos por octeto.

- **PLME-GET.confirm:** Esta función de callback contiene el valor del parámetro solicitado.

- **PLME-SET-TRX-STATE.request:** Solicita un cambio en el estado del transceptor a la capa física. El estado deseado puede ser: TRX_OFF para desactivar tanto el transmisor como el receptor, TX_ON para activar el transmisor y RX_ON para activar el receptor.

- **PLME-SET-TRX-STATE.confirm:** Respuesta a la solicitud de cambio de estado del transceptor.

- **PLME-SET.request:** Se trata de un "*Setter*" de los atributos de la capa física. Esta función solicita un cambio en el valor de uno de los atributos de la capa física.

- **PLME-SET.confirm:** Es el resultado de la solicitud de cambio de valor del atributo indicado.

5.2.4. FORMATO DEL PAQUETE PPDU

En este apartado se especifica el formato del paquete PPDU.

Por conveniencia, la estructura de paquete PPDU se presenta de modo que el campo de la izquierda será transmitido o recibido en primer lugar. Dentro de un campo del paquete se transmite o se recibe en primer lugar el bit menos significativo. Este mismo orden se aplica en la transmisión de las tramas entre la capa física y la MAC.

Cada paquete PPDU consta de los siguientes componentes básicos:

- Una cabecera de sincronización (SHR), que permite a un dispositivo receptor sincronizar y bloquear el flujo de bits transmitidos.

- Una cabecera que contiene información sobre la longitud de la trama.

- Una carga útil de longitud variable (PSDU), que se transporta a la capa MAC.

El formato del paquete PPDU se ilustra en la figura 8.

		Octets		
		1		variable
Preamble	SFD	Frame length (7 bits)	Reserved (1 bit)	PSDU
SHR		PHR		PHY payload

Figura 8. Formato del paquete PPDU. Imagen obtenida de "IEEE Std 802.15.4k™-2013"

5.2.5. DETECCIÓN DE ENERGÍA

La detección de energía está diseñada para ser usada por capas superiores como parte del algoritmo de selección de canal. Es una estimación de la potencia de la señal recibida en el ancho de banda del canal. No se utiliza para identificar o descodificar señales en el canal. El tiempo de medición es igual a 8 períodos de símbolo.

El resultado ED se informará a la MLME usando la función PLME-ED.confirm como un entero de 8 bits que van desde 0x00 a 0xFF. El valor mínimo indica que la potencia recibida es inferior a 10 dB por encima de la sensibilidad del receptor especificada.

5.2.6. INDICADOR DE CALIDAD DEL ENLACE

La medición es una caracterización de la fuerza y / o la calidad de un paquete recibido. La medición se puede implementar usando la medición de energía del canal (ED), una estimación de la relación señal a ruido o el uso de ambos métodos.

La medición de este parámetro se realiza para cada paquete recibido, y el resultado se comunica a la capa MAC utilizando PD-DATOS como un número entero comprendido entre 0x00 a 0xFF. Los valores mínimo y máximo del indicador de calidad del enlace (0x00 y 0xff) representan la calidad más baja y la más alta respectivamente y los valores intermedios están distribuidos de forma uniforme entre estos dos límites.

5.2.7. EVALUACIÓN DE LA OCUPACIÓN DEL CANAL

La capa física proporciona la capacidad de realizar una evaluación de ocupación del canal según uno de los tres siguientes métodos:

- **Modo 1**: Energía por encima del umbral. Informará un medio ocupado al detectar cualquier energía por encima del umbral de energía especificado utilizando la función de detección de energía del canal.

- **Modo2**: Detección de portadora única. Informará un medio ocupado únicamente ante la detección de una de señal de acuerdo con el estándar, con la misma modulación y características de la capa física que está actualmente en uso por el dispositivo. Esta señal puede estar por encima o por debajo del umbral de energía.

- **Modo 3**: Detección de portadora con la energía por encima del umbral. Informará un medio ocupado utilizando un combinación lógica de la detección de una señal con la modulación y características utilizadas por la capa física y de energía detectada en el canal por encima del umbral especificado , donde el operador lógico puede ser AND y OR.

Para cualquiera de los modos, si la función primitiva PLME-CCA.request es recibida por la capa física durante la recepción de un paquete PPDU, informará un medio ocupado.

La evaluación de canal ocupado se realiza atendiendo las siguientes especificaciones:

1. El umbral de energía equivale a 10 dB por encima de la sensibilidad del receptor especificada.

2. El tiempo de detección de la ocupación del canal es igual a 8 períodos de símbolo.

5.3. CAPA MAC

La capa MAC se encarga de controlar todos los accesos al canal de radio y es responsable de las siguientes tareas:

- Si el dispositivo es un coordinador, se encarga de generar balizas.
- Sincronización de balizas de red
- Apoyo a la asociación y desasociación de dispositivos
- Apoyo a la seguridad del dispositivo
- Emplear el mecanismo CSMA-CA para el acceso al canal
- Control y mantenimiento del mecanismo de GTS
- Proporcionar un vínculo fiable entre dos entidades MAC

La siguiente figura muestra el diagrama del modelo de referencia de la capa MAC.

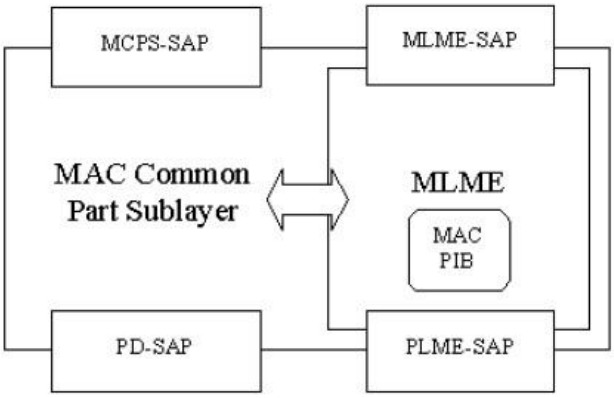

Figura 9. Modelo de referencia de la capa MAC. Imagen obtenida de "IEEE Std 802.15.4k™-2013"

5.3.1 SERVICIOS DE LA CAPA MAC

Al igual que la capa física, la capa MAC proporciona dos tipos de servicios, de datos y de gestión. Para comprender el funcionamiento del estándar, es importante conocer cómo funciona internamente por lo que es necesario conocer cuáles son y para qué sirven los servicios proporcionados por cada una de las capas ya que representan las líneas de actuación en el funcionamiento interno del mismo.

Los servicios de datos de la capa MAC son accesibles desde capas superiores a través de la entidad MCPS-SAP. Estos servicios, que se describen a continuación, son: MCPS-DATA.Request / Confirm / Indication y MCPS-PURGE. Request / Confirm.

- **MCPS-DATA.Request:** A través de esta función se solicita la transmisión de un paquete desde una capa superior a la MAC. Debe indicarse la dirección de origen, la dirección de destino, el identificador PAN de destino, la longitud de la carga útil, la carga útil a enviar y las opciones de envío: transmisión con ACK o transmisión sin ACK, transmisión GTS o transmisión CAP y transmisión directa o transmisión indirecta. La transmisión con o sin ACK indica si se espera o no el acuse de recibo del paquete por parte del destino, la transmisión GTS se realiza en el intervalo de tiempo reservado para las transmisiones del nodo en concreto, para utilizar este tipo de transmisión es necesario que la red sea balizada y la transmisión CAP se realiza en el tiempo de transmisión común para todos los dispositivos utilizando CSMA-CA.

- **MCPS-DATA.Confirm:** Indica el resultado de la solicitud de transmisión del paquete.

- **MCPS-DATA.Indication:** Cuando un paquete es recibido por otro dispositivo, la capa mac ejecuta esta función de callback que estará definida en una capa superior a través

de la que se puede conocer la carga útil, la dirección de origen y la calidad del enlace entre otros parámetros.

- **MCPS-PURGE.request:** Permite solicitar la eliminación de un paquete de la cola de transacciones de la capa MAC. Si se solicitó el envío de un paquete y esta transacción se encuentra aún en la cola, esta función permite eliminar dicha transacción para que el paquete no sea enviado.

- **MCPS-PURGE.confirm:** Esta función se ejecuta tras realizar una solicitud de eliminación de una transacción e indica si ha tenido éxito o no.

Los servicios de gestión de la capa MAC son accesibles desde capas superiores a través de la entidad MLME-SAP. En este caso, debido a que existe una gran cantidad de servicios de gestión de la capa MAC solo son definidos los más importantes.

- **MLME-SCAN:** Este servicio es utilizado por capas superiores para iniciar un barrido de una lista de canales. Un dispositivo puede utilizar este servicio para conocer el nivel de energía de unos canales determinados, para buscar al coordinador al que se encuentra asociado o para buscar a todos los coordinadores que están transmitiendo balizas.

- **MLME-ASSOCIATE:** Este servicio está formado por cuatro funciones diferentes que son ejecutadas durante el proceso de asociación de un nodo. Estas funciones son MLME-ASSOCIATE.Request / Indication / Response / Confirm.

- **MLME-DISASSOCIATE:** Este servicio está formado por cuatro funciones diferentes que son ejecutadas durante el proceso de desasociación de un nodo. Estas funciones son MLME-DISASSOCIATE.Request / Indication / Confirm.

- **MLME-SYNC:** Un dispositivo puede sincronizarse con el coordinador mediante este servicio que proporciona la capa MAC.

5.3.2. DESCRIPCIÓN FUNCIONAL

En los siguientes apartados se describen los mecanismos utilizados para iniciar y mantener una red, para asociar y desasociar nodos y para adquirir y mantener la sincronización entre los mismos y el coordinador.

Se realiza un escaneo de los canales para evaluar su estado actual, localizar todas las balizas en la zona o localizar una baliza en particular correspondiente a un nodo con el que ha perdido la sincronización. Antes de iniciar una nueva red, los resultados de una búsqueda de canales pueden ser utilizados para seleccionar un canal apropiado y la página del canal, así como un identificador de red que no está siendo utilizado por cualquier otra red en la zona. Tras una selección apropiada del canal junto con su identificador, un dispositivo FFD puede comenzar a funcionar como coordinador de la red.

El procedimiento de asociación describe las condiciones en las que un dispositivo puede unirse a una red y las condiciones necesarias para que un coordinador permita que se unan dispositivos. También se describe el procedimiento de desasociación, que puede iniciarse por el dispositivo asociado o por su coordinador.

Los mecanismos que permiten que los dispositivos adquirir y mantener la sincronización con un coordinador se describen en el apartado Se define el procedimiento de generación de balizas por parte del coordinador y la sincronización con los dispositivos asociados en una red balizada. Después se define el procedimiento de sincronización entre un coordinador y los dispositivos asociados en una red no balizada. También se describe un procedimiento para restablecer la comunicación entre un dispositivo y su coordinador tras haber perdido la sincronización.

INICIO Y MANTENIMIENTO DE LA RED

Un dispositivo ZigBee es programado para comenzar una búsqueda de canales a través de la función primitiva MLME-SCAN.request. Los canales son escaneados en orden desde el número de canal más bajo hasta el más alto. Durante la duración de la búsqueda, el dispositivo suspende las transmisiones de radiobalizas, en su caso, y sólo acepta tramas recibidas sobre los datos que sean relevantes para el análisis. Una vez que termina el escaneo de canales, el dispositivo reanuda las transmisiones de radiobalizas. Los resultados del escaneo se notifican a través de MLME-SCAN.confirm a las capas superiores.

BÚSQUEDA DE CANALES ACTIVA: Un escaneo activa permite a un dispositivo localizar cualquier trama del coordinador de la red. Esta función puede ser utilizada por un coordinador para seleccionar un identificador PAN antes de iniciar una nueva red, o puede ser utilizado por un dispositivo en el procedimiento de asociación.

Una exploración activa en un determinado conjunto de canales lógicos se solicita mediante el MLME-SCAN.request. Para cada canal lógico, el dispositivo cambia primero el canal en el que opera, y envía una trama de solicitud de baliza. Una vez que esta solicitud es transmitida, el receptor del dispositivo permanece activo durante el tiempo de escaneo especificado.

BÚSQUEDA DE CANALES PASIVA: Un escaneo pasivo, al igual que el activo, permite a un dispositivo para localizar cualquier coordinador que transmita tramas baliza en la frecuencia escaneada. En este caso, el comando de solicitud de baliza no se transmite. Este tipo de análisis puede utilizarse por un dispositivo final antes de la asociación. Durante un escaneo pasivo, la capa MAC descarta todas las tramas recibidas por el servicio de datos excepto las balizas. Se debe tener en cuenta que este tipo de escaneo solo puede utilizarse en redes balizadas, ya que no se envía la solicitud de envío de baliza.

Un escaneo pasivo de un determinado conjunto de canales lógicos se solicita mediante MLME-SCAN.request.

ASOCIACIÓN

Los dispositivos finales utilizan los mecanismos de escaneo de canales para detectar un coordinador de red ZigBee en la zona. Una vez que el dispositivo final ha detectado la presencia de un FFD, envía una solicitud de asociación al mismo utilizando MLME-ASSOCIATION.request. Las capas superiores del coordinador son notificadas de esta solicitud a través de MLME-ASSOCIATE.indication, por lo que evalúa la posibilidad de que el dispositivo se una y responde a través de MLME-ASSOCIATE.response. Esta respuesta no es enviada al dispositivo hasta que expire un tiempo de espera definido y el dispositivo hace polling al coordinador, es decir, solicita la transmisión de datos pendientes. Si MLME-ASSOCIATE.confirm tiene como valor SUCCESS, significará que el nodo se ha unido a la red.

La figura 10 muestra el procedimiento de asociación de un dispositivo.

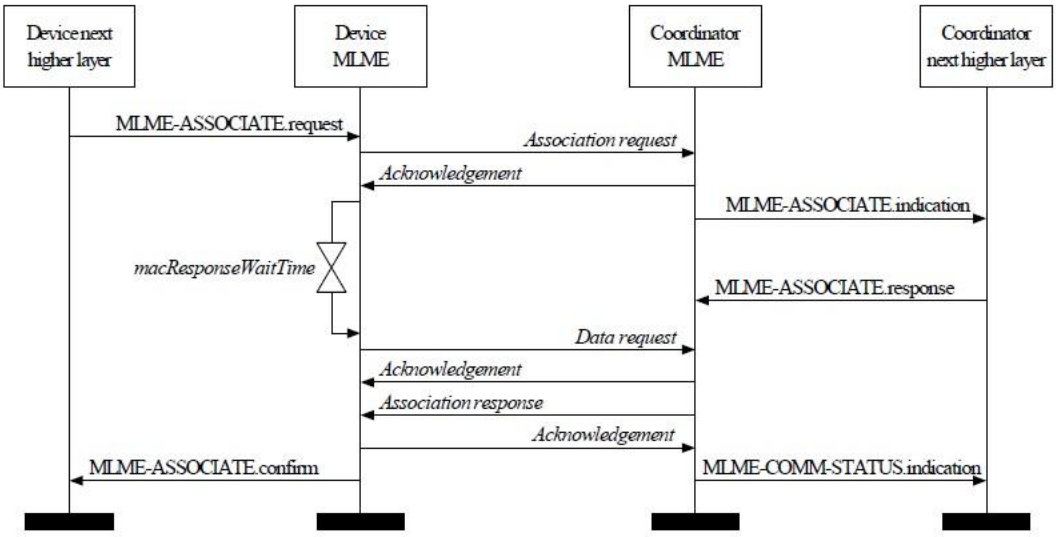

Figura 10. Procedimiento de asociación del protocolo ZigBee. Imagen obtenida de "IEEE Std 802.15.4k™-2013"

DESASOCIACIÓN

El procedimiento de desasociación puede ser iniciado tanto por un dispositivo final como por un nodo coordinador. El procedimiento es más sencillo que en la asociación, ya que en este caso no es necesario que el coordinador evalúe la posibilidad de desasociación. El proceso comienza desde las capas superiores haciendo uso de MLME-DISASSOCIATE.request que se traducirá en el envío de una trama con la notificación de desasociación. Una vez que la trama llega al dispositivo de destino, el contenido de la trama pasa de la capa física a la MAC y la entidad de gestión confirma la desasociación y envía un ACK al nodo de origen. Las capas

superiores del nodo de destino son notificadas del evento de desasociación a través de MLME-DISASSOCIATE.request. La figura 11 muestra un diagrama del proceso de desasociación iniciado por un dispositivo final.

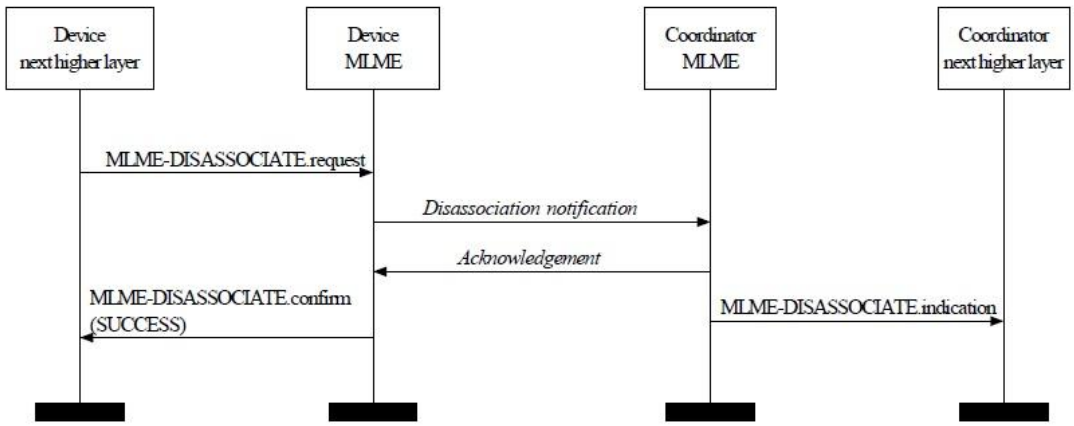

Figura 11. Procedimiento de desasociación del protocolo ZigBee. Imagen obtenida de "IEEE Std 802.15.4k™-2013"

SINCRONIZACIÓN

Dependiendo del tipo de red el procedimiento para que un dispositivo final permanezca sincronizado con su coordinador cambia. Los nodos coordinadores pueden transmitir unas señales guía llamadas balizas que el resto de dispositivos utiliza para sincronizarse con él.

En el caso de una red balizada, el coordinador envía constantemente balizas que indican el comienzo de la supertrama. Una supertrama es como se le llama al conjunto de intervalos temporales que comprenden un ciclo de transmisión en la red. Este ciclo de transmisión está formado por el CAP en el que todos los dispositivos pueden transmitir datos de forma directa o indirecta utilizando el mecanismo de acceso al canal CSMA-CA, por el CFP en el que cada dispositivo dispone de un slot de tiempo garantizado para poder transmitir y un intervalo de tiempo inactivo en el que los dispositivos se encuentran dormidos. La figura 12 muestra el formato de una supertrama.

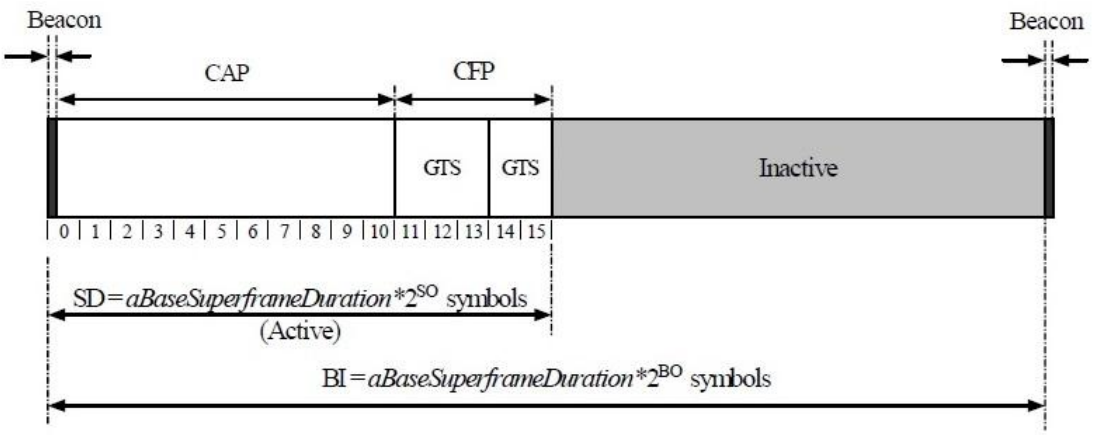

Figura 12. Formato de una supertrama. Imagen obtenida de "IEEE Std 802.15.4k™-2013".

Los dispositivos escuchan estas balizas para permanecer sincronizados con el coordinador, en las redes balizadas la sincronización es continua y se deben respetar los intervalos de tiempo de transmisión e inactivos.

En redes no balizadas la sincronización es más sencilla, no existen unos intervalos de tiempo prefijados para transmitir o permanecer dormidos, sino que cada dispositivo puede hacer poll al coordinador para iniciar una comunicación.

5.4. CARACTERIZACIÓN DE LA RED

En este apartado se describen las características de la red que va a ser implementada en este proyecto, indicando el tipo, la topología junto con una descripción funcional. Las características de la red han sido definidas de tal modo que optimice la funcionalidad que queremos conseguir con la aplicación, reduciendo la complejidad y el consumo de energía lo máximo posible.

5.4.1. TIPO

La aplicación que va a ser desarrollada consiste en el envío de una serie de tramas desde los dispositivos finales al coordinador. Los nodos finales transmitirán información sobre la temperatura actual en los mismos, por lo que el tiempo de envío entre tramas deberá ajustarse al ritmo de cambio de esta variable ambiental.

El sensor de temperatura que incorporan los nodos tiene un ritmo de cambio de aproximadamente un grado centígrado por minuto, lo que hace inútil enviar esta información

con elevada frecuencia. Dicho esto, se puede suponer que el tráfico de la red será muy bajo por lo que no es necesario utilizar slots de tiempo garantizados para transmitir, función que ofrecen las redes balizadas, ya que se podrá acceder sin problemas al canal. Además, los datos de temperatura no son tan críticos como para necesitar garantizar el envío de los mismos como puede ser en el caso de un control automático industrial, por ejemplo.

El coordinador deberá estar siempre despierto por el hecho de que incorpora un LCD que estará activo en todo momento, por lo que el período inactivo de las supertramas de una red balizada no podría ser aprovechado por el coordinador.

Teniendo en cuenta los aspectos comentados en los párrafos anteriores, para reducir la complejidad en la sincronización y dado que la aplicación no se ajusta a las funciones de una red balizada se ha decidido realizar una red no balizada, permitiendo a los dispositivos finales dormir el tiempo que estimen necesario entre envíos de tramas.

5.4.2. TOPOLOGÍA

El protocolo ZigBee está diseñado para auto-organizarse según se van añadiendo dispositivos a la red. Esto implica que el proceso de formación de la topología de red es autónomo por lo que no es necesario realizar ningún tipo de configuración. Dependiendo del tipo de topología elegido se establece una estrategia de creación de enlaces entre los nodos.

Una red ZigBee puede adoptar una topología en malla, en árbol o en estrella. La topología en malla permite la comunicación directa entre cualquier par de nodos de la red, siempre que se encuentren dentro del rango de cobertura o a través de uno o varios routers que permitan extender el rango de la red. Cuando un dispositivo se incorpora a la red, si se trata de un router se crean enlaces con todos los routers próximos y si se trata de un dispositivo final se crea un enlace directo con el router cuya ruta hacia el coordinador sea la de menor coste. De esta forma se consigue formar una red mallada, reduciendo el número de saltos en la comunicación entre dispositivos.

Una red con topología en árbol permite dividir la red en niveles de profundidad. Reduce el número de saltos en la comunicación entre dispositivos con respecto a una red con topología en estrella salvo en el peor caso en el que la transmisión es direccionada a través del nodo raíz (el coordinador). Cuando un dispositivo se une a la red, se crea un enlace con el router cuya ruta hacia el coordinador es la menos costosa, se trate de un router o un dispositivo final.

En este caso concreto, la aplicación que va a ser desarrollada requiere de comunicación directa entre los dispositivos finales y el coordinador, nunca entre dispositivos finales. Por este motivo resulta más útil formar una red con topología en estrella en la que cualquier dispositivo pueda comunicarse con el coordinador a través de la ruta de menor coste, ya sea de forma directa o a través de uno o más routers.

En la siguiente figura se muestran las distintas topologías que puede adoptar una red ZigBee.

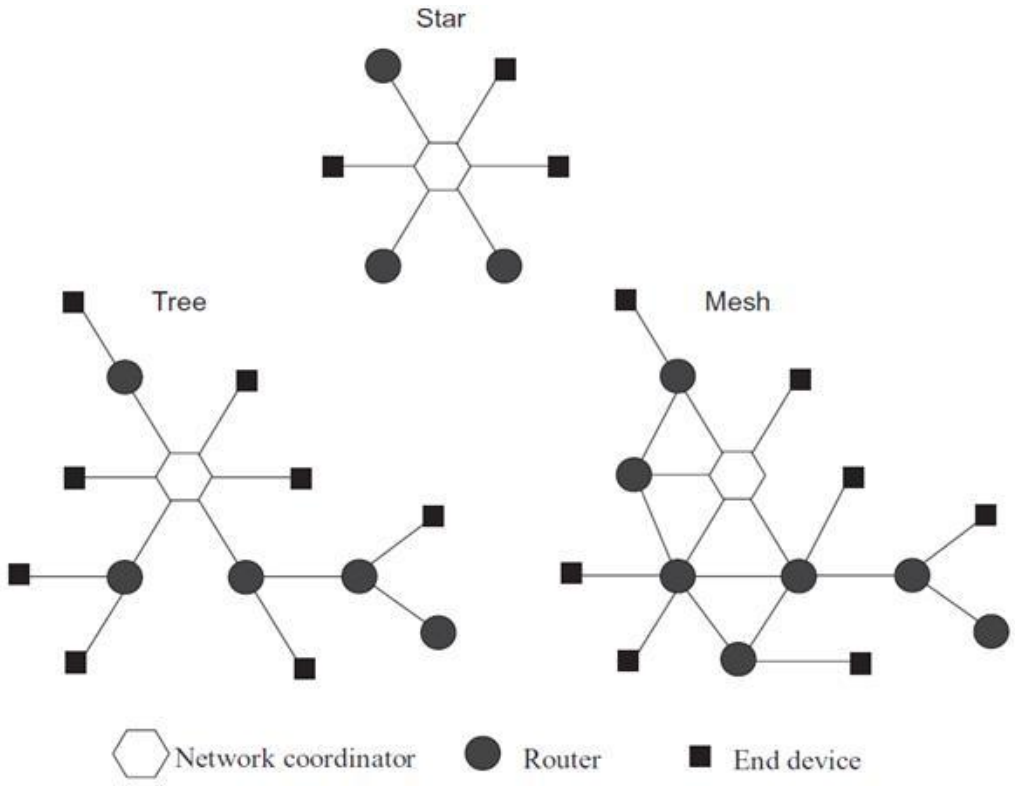

Figura 13. Posibles topologías de una red ZigBee.

5.4.3 DESCRIPCIÓN FUNCIONAL

Como se ha indicado en los apartados anteriores, se ha decidido implementar una red ZigBee no balizada y con una topología de estrella. Dicha red se organiza de forma autónoma comenzando por la creación de la red por parte del coordinador. En primer lugar, el coordinador de la red realiza un escaneo de los 27 canales disponibles para cada una de las 32 páginas de canales. Mediante este proceso, determina que canal es el más óptimo en función de la estimación de la relación señal a ruido que realiza. Una vez que ha decidido el canal que va a ser utilizado, el coordinador espera a que algún dispositivo le envíe una solicitud de envío de baliza, por lo que no hará nada hasta este momento. Esta solicitud de envío de baliza será transmitida por un dispositivo que quiere unirse a la red en el período de escaneo para detectar a los coordinadores que se encuentran en el rango de cobertura.

Una vez que un dispositivo final encuentra a un coordinador, éste le envía una solicitud de asociación que deberá ser confirmada por el coordinador asignando una dirección de red que

no se encuentre en uso al nuevo dispositivo. Los dispositivos finales se encuentran la mayor parte del tiempo en modo sleep, por lo que una vez finalizado el período de asociación a la

red, éstos entrarán en modo de bajo consumo durante un intervalo de 30 segundos, momento en el que despiertan y envían los datos de temperatura al coordinador.

La transmisión de datos se realiza de forma directa, es decir, un dispositivo que quiera transmitir una trama al coordinador podrá hacerlo una vez que el canal se encuentre libre y no tendrá que esperar a que el coordinador haga poll solicitando el envío de datos pendientes.

Para ofrecer una interfaz visual de los eventos de asociación y de transmisión de tramas de datos se utiliza un led en cada uno de los dispositivos. En el coordinador, este led producirá un parpadeo durante un corto intervalo de tiempo cada vez que un nodo se asocie a la red y en los dispositivos finales permanece encendido mientras están despiertos y apagado cuando están dormidos, lo que significa que se transmite una trama cuando se enciende el led.

6. TECNOLOGÍA

6.1. ELECCIÓN

En los párrafos que siguen se va a realizar una comparación entre las distintas opciones comerciales para desarrollar este proyecto, que ha servido para diferenciar las tecnologías y poder tomar una decisión acertada.

Aparte de los módulos que han sido descritos en el apartado 4.1, se ha evaluado la posibilidad de utilizar para desarrollar el proyecto la plataforma de evaluación ATMEGA256RFR2 Xplained Pro de la compañía ATMEL. Este tipo de plataformas están diseñadas para poder realizar una evaluación real del microcontrolador así como para desarrollar proyectos de prototipado en los que el objetivo no es utilizar la placa para producir un producto final, sino para conseguir realizar de forma más sencilla y menos costosa un prototipo del proyecto.

La tabla 4 muestra una comparación entre las diversas tecnologías.

NOMBRE	PRECIO	ARQUITECTURA	RAM	FLASH
XBee – DIGI INTERNATIONAL	45€ ud	HCS08 8-bits	2 KB	32 KB
ETRX357 – TELEGESIS	15.50€ ud	ARM 8-bits	12KB	192KB
ATZB-S1-256-3-0-C – ATMEL	28.63€ ud	AVR 8-bits	32KB	256KB
JN5139-001-M00 – NXP	25.75€ ud	ARM 32-bits	96KB	192KB
ATMEGA256RFR2 XPLAINED PRO – ATMEL	**32.20€ ud**	**AVR 8-bits**	**32KB**	**256KB**

SENSOR DE TEMPERATURA	DEPURADOR/ PROGRAMADOR	PRECIO TOTAL (3 ud)
2.30€ ud	INCLUIDO EN EL KIT	141.90€
2.30€ ud	112.30 €	165.70€
2.30€ ud	87.35€	180.14€
2.30€ ud	315€	399.15€
INTEGRADO EN LA PLACA	**INTEGRADO EN LA PLACA**	**96.6€**

Tabla 4. Comparación entre las distintas tecnologías candidatas para el desarrollo del proyecto.

Como se puede comprobar en la tabla, la plataforma de evaluación de Atmel tiene la mejor relación calidad precio, además de incorporar una serie de componentes que el resto de las placas no tienen. De este modo, se ha tomado la decisión de utilizar la plataforma ATMEGA256RFR2 XPLAINED PRO para desarrollar el proyecto.

6.2. DESCRIPCIÓN DE ATMEGA256RFR2 XPLAINED PRO

Figura 14. Atmega256RFR2 XPLAINED PRO

El kit de evaluación de Atmel ATmega256RFR2 Xplained Pro es una plataforma de hardware que integra los componentes necesarios para desarrollar el proyecto como conectores con los pines de I/O, una interfaz USB con depurador integrado en la placa, un sensor de temperatura, dos botones mecánicos, dos cristales de 16MH y 32KHz y una antena reduciendo así el tiempo de desarrollo.

El microcontrolador ATmega256RFR2 que integra la plataforma puede programarse realizando una descarga de Flash a través de la interfaz USB conectándola directamente con un PC. Para ello utiliza el depurador integrado que incorpora, con el que también es posible depurar el programa mediante el entorno de desarrollo Atmel Studio 6.2.

La plataforma incorpora los siguientes componentes:

- Microcontrolador ATmega256RFR2 de Atmel
- Depurador integrado
- Interfaz USB

- Programación y depuración a través de la interfaz JTAG
- Virtual COM-port Interface via UART
- Data Gateway Interface via SPI o TWI
- I/O Digitales
- Dos botones mecánicos (Usuario y Reset)
- Un Led de usuario
- Cinco conectores de extensión
- Una antena de cerámica
- Un conector SMA para conectar una antena externa
- Sensor de temperatura y EEPROM
- Dos Fuentes de alimentación posibles
- Alimentación externa
- Cristal de 16MHz
- Cristal de 32kHz

En la figura 15 se pueden identificar los componentes que incorpora la plataforma.

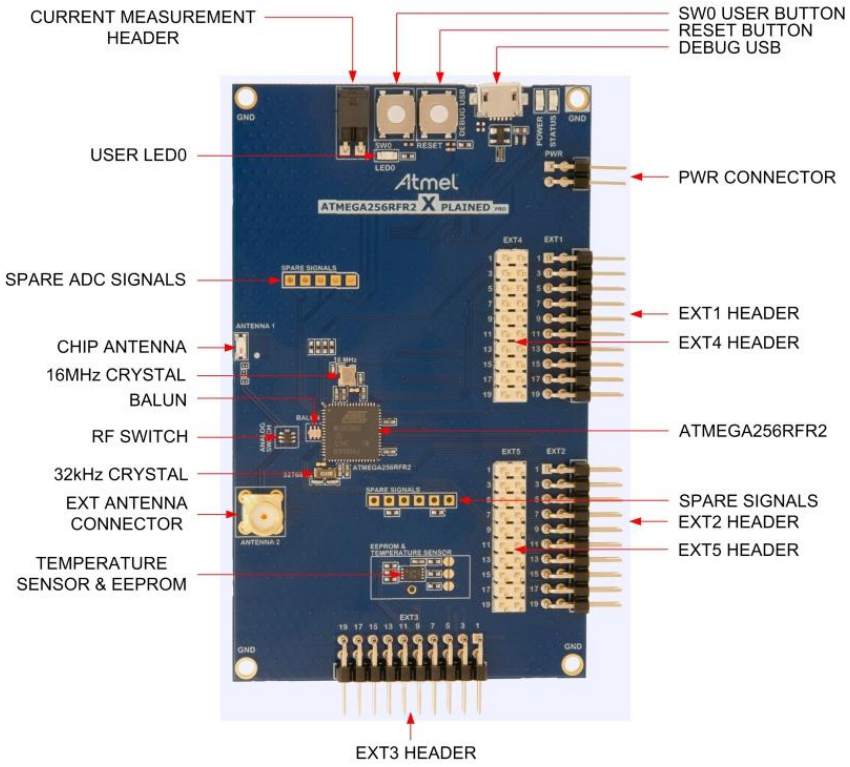

Figura 15. Componentes de la plataforma ATmega256RFR2 Xplained Pro

La tensión de alimentación de la plataforma depende de la fuente de alimentación utilizada, puede utilizar un PC como fuente de alimentación a través del depurador USB o una fuente extena a través de los pines etiquetados en la figura x. como PWR CONNECTOR. En el caso de utilizar una fuente de alimentación conectada a estos pines, es necesario proporcionar dos tensiones, 5V y 3V3. La corriente mínima recomendada que debe poder proporcionar la fuente de alimentación es de 500mA y la corriente máxima especificada por seguridad es de 2A. La plataforma detecta automáticamente las fuentes de tensión que están conectadas y elige cual utiliza en el siguiente orden:

1. Alimentación externa

2. Depurador USB integrado

El diagrama de bloques de la plataforma junto con el esquemático del microcontrolador, del depurador integrado, de la antena y del sensor de temperatura se encuentran el el ANEXO 15.3.

6.2.1. MICROCONTROLADOR

El ATmega256RFR2 es un microcontrolador de 8-bits fabricado por la empresa Atmel basado en la arquitectura AVR que incorpora un transceptor para la banda ISM de 2.4GHz adecuado para el protocolo ZigBee. Posee un rendimiento de hasta 1 MIPS por MHz, pudiendo alcanzar los 16 MIPS. El rendimiento del MCU es configurable para obtener así la relación velocidad – consumo óptima para cada caso concreto.

Atmel proporciona una serie de bibliotecas compatibles para este microcontrolador, que pueden ser de utilidad sobre todo en el caso de drivers para los módulos hardware. En los siguientes epígrafes se describen sus principales características así como los módulos hardware que incorpora. La figura 16 muestra el diagrama de bloques del microcontrolador.

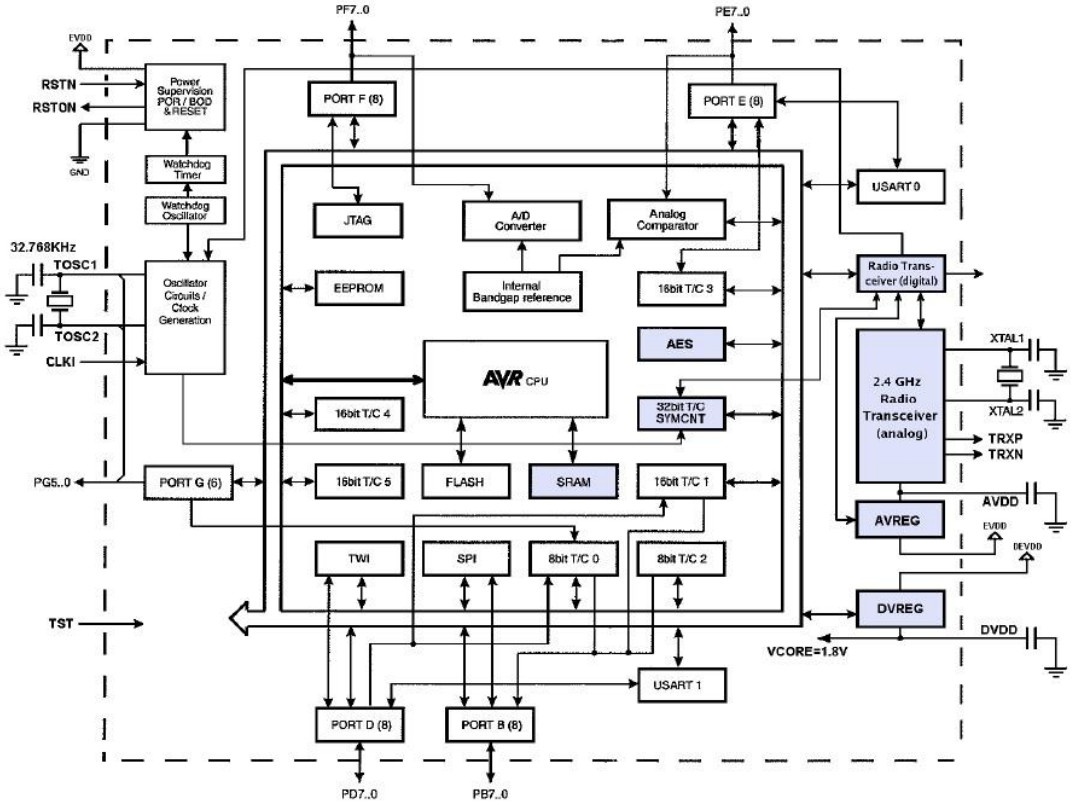

Figura 16. Diagrama de bloques del microcontrolador.

CARACTERÍSTICAS

Las características del microcontrolador de Atmel, ATmega256RFR2 son enumeradas a continuación.

- Arquitectura RISC:
  - 135 instrucciones
  - 32x8 registros de propósito general
  - Rendimiento de hasta 16 MIPS @16 MHz - 1.8V
- Memoria de datos y programa:
  - 256Kbytes de Flash
  - 8KBytes de EEPROM
  - 32KBytes de SRAM
- Interfaz JTAG
- Periféricos:
  - Multiples temporizadores/contadores y canales PWM
  - Contador en tiempo real con oscilador independiente
  - Conversor analógico/digital 10-bit, 330 ks/s
  - Comparador analógico
  - Interfaz SPI Maestro/Esclavo
  - Dos USART programables
  - Interfaz TWI
- Temporizador Watchdog con oscilador independiente
- Transceptor de bajo consumo para la banda ISM de 2.4 GHz
  - Amplificador de potencia
  - Tasas de transmisión: 250 kb/s, 500 kb/s, 1 Mb/s y 2 Mb/s
  - Sensibilidad del receptor: -100 dBm; Potencia de transmisión: 3.5 dBm
  - Contador de símbolos de 32 Bits
  - Detección SFD, ensanchado; desensanchado ; CRC-16
  - Diversidad de antena
  - Buffer de 128 bytes en TX / RX

- Detector de fase

- Sintetizador PLL con espaciado de canales de 5 MHz y 500 kHz para la banda ISM de 2.4 GHz
- Generación de encriptación AES mediante hardware
- 38 pines de entrada y salida
- Grado de velocidad: 0 – 16 MHz para 1.8 – 3.6V

CONSUMO DE ENERGÍA

Uno de los aspectos más relevantes a tener en cuenta son los diferentes modos de ahorro de energía que posee el microcontrolador, ya que en una red ZigBee los nodos transmiten pequeñas cantidades de datos y no se requiere que la CPU este activa todo el tiempo por lo que se utilizan los modos de sleep para mantener dormido a los nodos durante la mayor parte del tiempo optimizando el consumo energético.

La figura 17 ilustra un diagrama del consumo de corriente del microcontrolador y el transceptor conjuntamente en los estados más reconocibles:

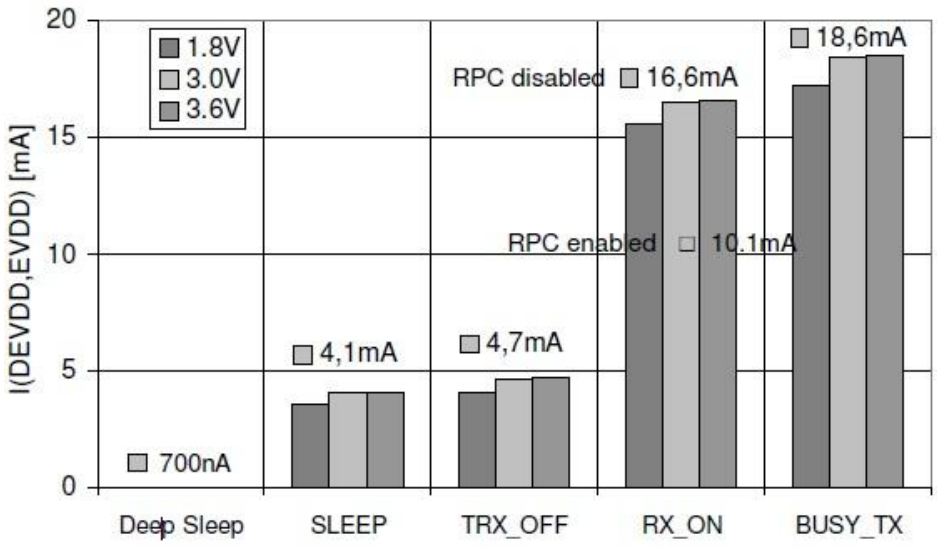

Figura 17. Consumo conjunto del microcontrolador y el transceptor en los estados más reconocibles.

En la figura 18 aparecen los estados más comunes en los que el microcontrolador puede encontrarse, cuyo significado se define a continuación:

- ***Deep Sleep***: en este modo el consumo del microcontrolador es menor a los 700nA. El MCU entra en este modo cuando se encuentra en modo "power-save" o "power-down" y el transceptor está desactivado.

- ***SLEEP***: El estado sleep tiene varios modos. Cada uno de ellos tiene una finalidad diferente, por lo que difieren en los clocks y osciladores que permanecen activos mientras se encuentre en ese estado como también las fuentes que pueden despertarlo. La figura x. muestra una tabla con los diferentes modos de sleep.

- **TRX_OFF**: este es el estado en el que se encuentra el microcontrolador cuando el transceptor está desactivado.

- **RX_ON**: estado que adopta el MCU mientras se encuentra transmitiendo datos.

- **BUSY_TX**: estado que adopta el MCU mientras se encuentra recibiendo datos.

	Active Clock Domains					Oscillators		Wake-up Sources								
Sleep Mode	clkCPU	clkFLASH	clkIO	clkADC	clkASY	Main Clock-source Enabled	Timer Oscillator Enabled	INT7:0 and Pin Change	TWI Address Match	Timer/Counter2	SPM/EEPROM Ready	ADC	WDT Interrupt	Other I/O	Symbol Counter	Transceiver
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	X(4)	X(4)
ADCNRM				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		X(4)	X(4)
Power-down								X(3)	X				X		X(4)	X(4)
Power-save					X		X(2)	X(3)	X	X			X		X(4)	X(4)
Standby (1)						X		X(3)	X				X		X(4)	X(4)
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X(4)	X(4)

Figura 18. Relojes activos y fuentes de activación (*wake-up*) en los diferentes modos de *sleep*.

ARQUITECTURA

En este apartado se describe la arquitectura del núcleo AVR en general. La función principal de la CPU es asegurar la correcta ejecución del programa. Por lo tanto, debe ser capaz de acceder a las memorias, realizar cálculos, controlar los periféricos y gestionar las interrupciones. La figura 19 muestra el diagrama de bloques de la arquitectura del núcleo.

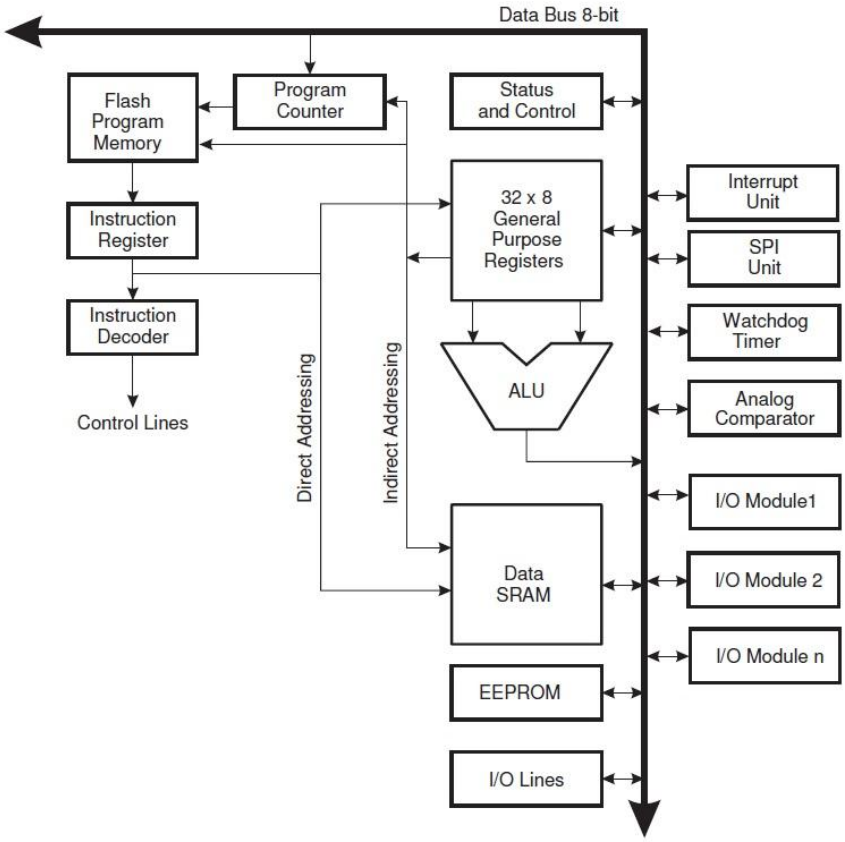

Figura 19. Diagrama de bloques de la arquitectura AVR.

Para mejorar el rendimiento y el paralelismo, AVR utiliza una arquitectura Hardvard, separando las memorias y los buses para datos y programa. Las instrucciones en la memoria de programa son ejecutadas con un único nivel de canalización. Mientras una instrucción está siendo ejecutada, la siguiente instrucción es buscada en la memoria de programa. Este concepto permite ejecutar instrucciones en cada ciclo de reloj, lo que mejora el rendimiento.

El archivo de registro contiene 32 registros de 8 bits de propósito general con un tiempo de acceso equivalente a un único ciclo de reloj, lo que permite una operación de la ALU por cada ciclo de reloj. En una operación típica de la ALU, dos operandos son emitidos desde el archivo de registro, la operación es ejecutada y el resultado es almacenado en el archivo de registros, en un único ciclo de reloj.

Seis de los 32 registros pueden ser utilizados como tres punteros de direcciones de 16 bits para direccionamiento de registros de datos, lo que proporciona un direccionamiento más eficiente debido a que los registros de direcciones de 16 bits permiten direccionar hasta 65535 bytes frente a los 255 bytes posibles con un registro de 8 bits. La no necesidad de guardar punteros de direccionamiento en varios registros de 8 bits (típico 2 registros de 8 bits para la dirección y 1 registro más para el orden en el que deben ir los registros) permite que el proceso de direccionamiento sea mucho más rápido.

El módulo de interrupciones tiene sus registros de control en el espacio de memoria de I/O junto con un bit adicional para habilitar las interrupciones de forma global en el registro de estado. Todas las interrupciones tienen vectores de interrupciones separados en la tabla de interrupciones. La prioridad de las mismas viene determinada por la posición que ocupan en la tabla. El vector de interrupción con la dirección más baja tiene la mayor prioridad (RESET). Hasta 58 fuentes de interrupción diferentes pueden ser controladas por el módulo de interrupciones.

MEMORIA

La memoria en el microcontrolador ATmega256RFR2 se divide en dos grupos, la memoria de datos y la memoria de programa.

La figura 20 muestra como está organizada la memoria de datos en el ATmega256RFR2. Las primeras posiciones de memoria están destinadas a los 32 registros de propósito general que van de 0000h – 001Fh, las siguientes posiciones que van de 0020h – 005Fh están asignadas para la memoria de I/O, seguidas de un espacio para la memoria de I/O extendida y finalmente se encuentra la memoria de datos interna SRAM de 32KBytes. Esta memoria es de lectura/escritura por lo que puede ser modificada durante la ejecución del programa, tanto los registros como las variables almacenadas en la SRAM. Todas las variables que declaremos en el programa son guardadas en esta memoria, de modo que 32KBytes son más que suficientes si hablamos de software embebido.

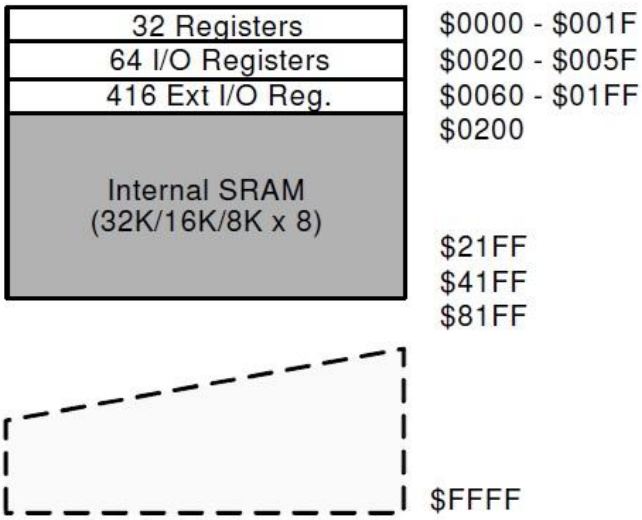

Figura 20. mapa de memoria de datos

La memoria Flash de programa está dividida en dos secciones, una sección destinada al programa de aplicación y otra destinada a proporcionar la capacidad de auto-programación. Esta separación es realizada por seguridad. El espacio de memoria destinado al programa de aplicación está dividida en 3 páginas que almacena el código y las variables constantes del programa de aplicación. Es preferible utilizar la memoria no paginada para obtener mayor rendimiento en la ejecución debido a que el direccionamiento ocasiona retardo. La figura 21 muestra el mapa de memoria.

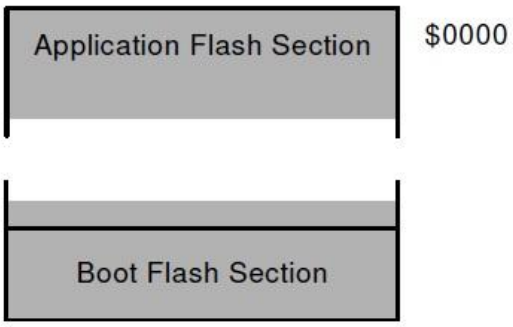

Figura 21. mapa de memoria de programa

6.2.2. SENSOR DE TEMPERATURA

El AT30TSE758 es un sensor de temperatura digital fabricado por Atmel capaz de medir temperaturas desde -55ºC hasta 125ºC. Incorpora unos registros no volátiles a través de los cuales es posible configurar el sensor y obtener los datos de temperatura con una resolución de 9-12 bits que equivale a una resolución de temperatura de 0.50000ºC-0.0625ºC.

La figura 22 muestra el diagrama de bloques del integrado.

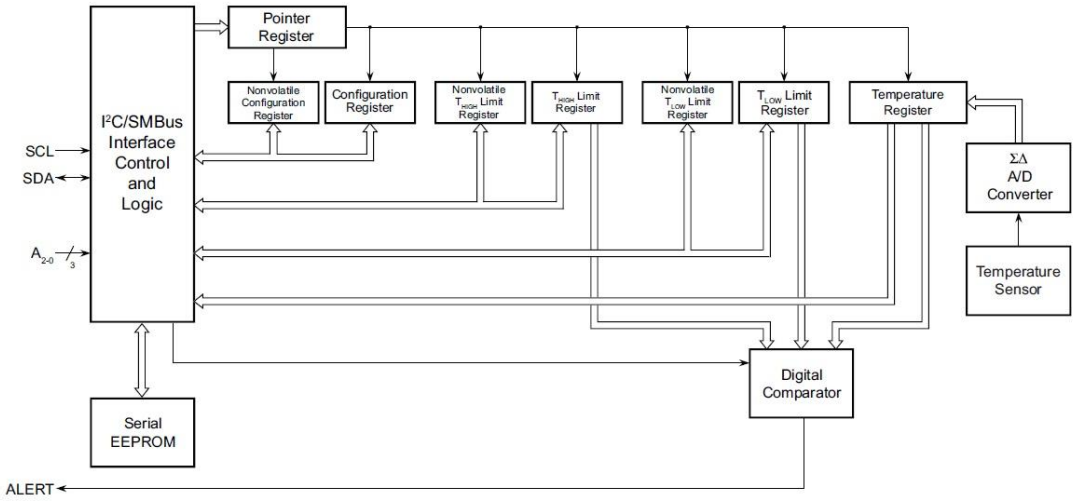

Figura 22. Diagrama de bloques del sensor de temperatura digital AT30TSE758.

El integrado utiliza los protocolos de comunicación I2C / TWI y SMBus para establecer comunicación con el microcontrolador. La dirección de esclavo del sensor para el protocolo TWI está determinada por los pines A2 A1 y A0 dependiendo de si se encuentran a nivel alto o a nivel bajo. El protocolo TWI será el utilizado para la comunicación con el microcontrolador ya que es el único que incorpora de los tres mencionados. Este protocolo está basado en una comunicación Maestro/Esclavo a través de dos hilos, uno de ellos es utilizado para transmitir la señal de datos (SDA) y el otro para transmitir la señal de reloj que sincroniza ambos dispositivos (SCL).

La comunicación sigue el siguiente proceso: en primer lugar el microcontrolador que actua como maestro en la comunicación envía una señal de “start” indicando al resto de dispositivos conectados al bus que quiere iniciar una comunicación seguida de el byte de dirección. El byte de dirección contiene la dirección del esclavo con el que se quiere comunicar, en este caso el sensor de temperatura, que ocupa los 7 bits más significativos. El bit restante indica si la operación que se quiere realizar es de lectura o escritura. El valor 0 en este bit indica que el Maestro quiere escribir datos en un registro del sensor, por el contrario, si vale 1 indica que quiere leer datos de los registros del sensor.

Es importante saber que la operación de lectura o escritura se realizará en el registro del sensor al que apunte el puntero de registro. El puntero de registro puede ser modificado mediante una operación de escritura en el mismo. Para ello, el microcontrolador deberá enviar la señal de start seguida del byte de dirección con el valor 0 en el último bit (para indicar que la operación es de escritura). Una vez recibido el ACK por parte del esclavo, el próximo byte que envíe será siempre escrito en el registro del puntero del sensor. Una vez modificado el puntero, es posible leer tantas veces como se desee el registro al que apunta el puntero. En las operaciones de escritura, el primer byte de datos siempre modificara el valor del puntero, para

así poder cambiar siempre que se desee el registro en el que se realizan las operaciones de lectura/escritura.

Tanto en la operación de lectura como en la de escritura, el primer registro en el que se lee o escribe es el que es apuntado por el puntero de registro. Las siguientes operaciones, dentro de una misma comunicación, se realizaran en los registros consecutivos hasta que finalice la comunicación mediante el envío de la señal de stop por parte del maestro, en este caso, el microcontrolador. Dos son los registros que almacenan los datos de temperatura sensados, por lo que se deben leer 2 bytes.

6.2.3. DEPURADOR/PROGRAMADOR EMBEBIDO

El depurador integrado (EDBG) es un dispositivo USB compuesto de 3 interfaces; un depurador, un puerto de comunicación virtual (Virtual COM Port) y una interfaz de transmisión de datos (DGI).

Junto con el programa Atmel Studio, se puede programar y depurar el ATmega256RFR2 a través de la interfaz de depuración. En la plataforma, la interfaz JTAG está situada entre el depurador embebido (EDBG) y el microcontrolador.

El puerto virtual de comunicación está conectado a un puerto UART del ATmega256RFR2 y proporciona una manera sencilla de comunicarse con la aplicación de destino en un PC utilizando el terminal. Ofrece una configuración de la velocidad en baudios, paridad y bits de parada variables. Hay que tener en cuenta que la configuración de la UART del microcontrolador debe coincidir con los ajustes indicados en el terminal del PC.

La DGI está formada por varias interfaces de datos físicos para establecer comunicación con un PC. La comunicación a través de las interfaces son bidireccionales. Puede ser utilizado para enviar eventos, estados y variables en tiempo real del programa, o como un canal de datos de estilo printf() genérico. Esta interfaz resulta muy útil para la depuración.

6.3. HERRAMIENTAS DE DESARROLLO

6.3.1. ATMEL STUDIO

AtmelStudio 6.2 es una plataforma de desarrollo integrado utilizada en este proyecto para desarrollar el código en el lenguaje de programación C de la aplicación, así como compilarlo, depurarlo y programarlo. El programa permite depurar y programar las aplicaciones en los microcontroladores de Atmel basados en ARM Cortex-M y AVR. Además de C también permite compilar código en C++ y ensamblador.

El programa cuenta con multitud de herramientas integradas para facilitar la depuración, así como un conjunto de bibliotecas comprendidas en “atmel software framework”. Es capaz de detectar la plataforma ATmega256RFR2 XPLAINED PRO, por lo que permite visualizar todos los registros del microcontrolador durante la depuración de un programa. Esta herramienta es extremadamente útil para depurar drivers o código dependiente de hardware. Indica el estado actual de todos los registros en el momento en el que se detiene la depuración, mostrando en rojo los registros que han cambiado con respecto al valor que tenía previamente.

La figura 23 muestra el formato de la ventana inicial del programa, donde puede verse una ventana llamada “start page” en laque aparecen los proyectos recientes que han sido utilizados así como accesos directos para crear un nuevo proyecto, crear un nuevo proyecto de ejemplo o abrir proyecto.

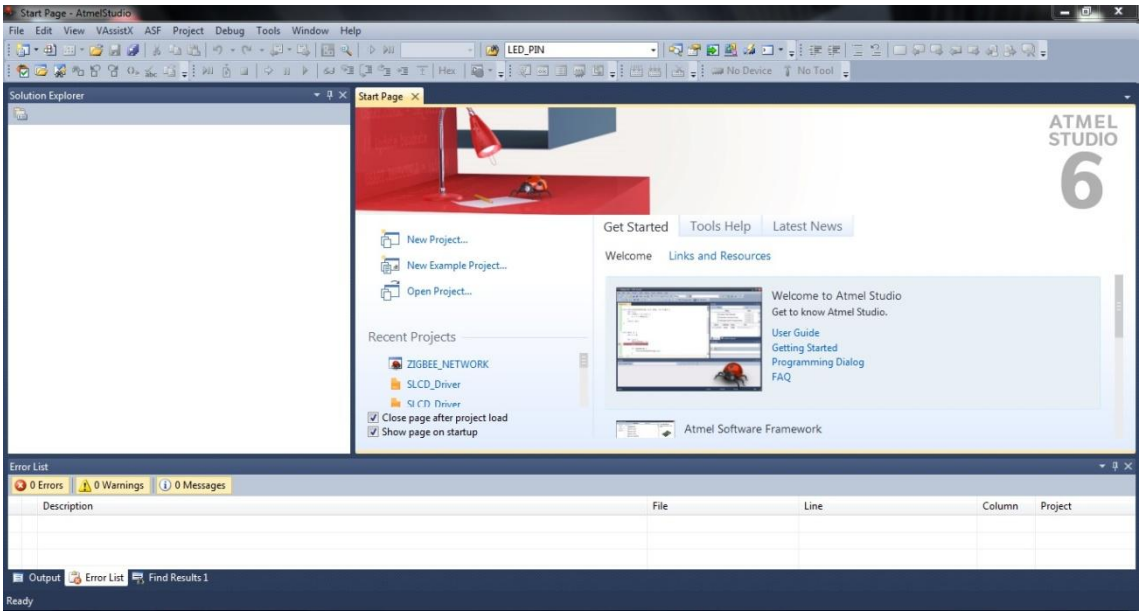

Figura 23. Formato de la ventana inicial del programa Atmel Studio 6.2.

Como se puede ver en la imagen, entre las pestañas que se sitúan en la parte superior podemos encontrar la pestaña ASF, la cual sirve para explorar “atmel software framework” y seleccionar las bibliotecas que se desean utilizar para el desarrollo.

La pestaña Debug, contiene todas las utilidades referentes al control de la depuración, la mayoría de ellas accesibles desde la barra de herramientas de acceso rápido del depurador. Para depurar el programa, se suele utilizar la opción “start debugging and break” que como su propio nombre indica, lo que hace es iniciar la depuración y detenerla en la primera línea del programa. Una vez iniciada la depuración se pueden visualizar las variables locales del ámbito en el que se encuentra el programa, las variables globales añadidas a un panel de visualización a través de la acción “add watch”, los valores almacenados en RAM, los registros del microcontrolador, así como realizar acciones para facilitar la depuración del programa: step over para pasar a la siguiente línea de código, step out para salir del ámbito de una función, step into para entrar al ámbito de una función, break para detener la depuración, start debugging para continuar con la misma o stop debugging para finalizarla.

La programación del dispositivo se realiza a través de “device programming” que se encuentra en la pestaña “Tools”. Mediante esta ventana podemos programar o realizar una simulación de un microcontrolador. Para realizar una descarga del programa de aplicación se debe seleccionar la herramienta que queremos programar y después, en la pestaña de “memorias” seleccionar la acción “program” referida a la memoria Flash. Se puede seleccionar si se borra y si se verifica la memoria antes de programarla con los checkboxs “Erase Flash Before Programming” y “Verify Flash Before Programming” respectivamente. La figura 24 muestra la apariencia de esta ventana.

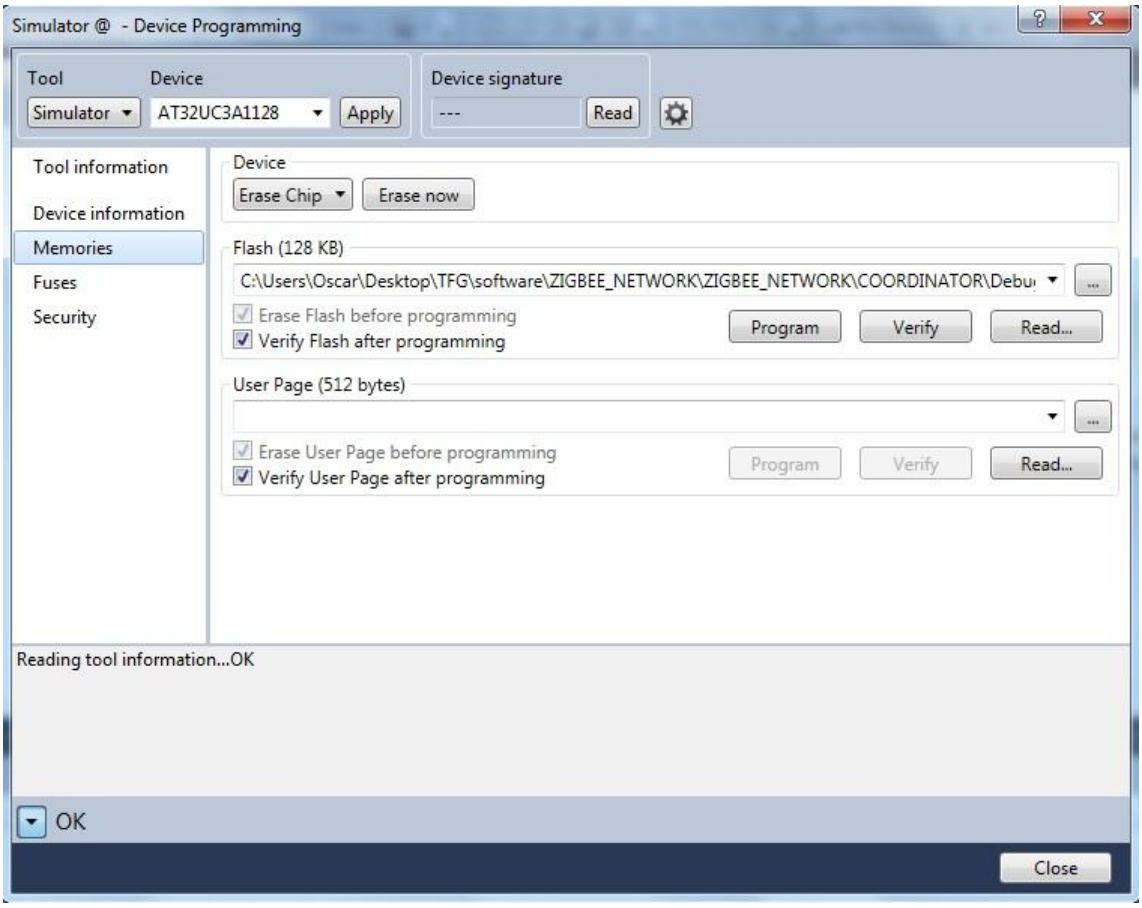

Figura 24. Apariencia de la ventana de programación de dispositivos.

6.3.2. ATMEL SOFTWARE FRAMEWORK

Atmel SoftwareFramework es una herramienta integrada en Atmel Studio a través de la cual se tiene acceso al conjunto de bibliotecas desarrolladas por Atmel. Para tener acceso a esta herramienta, el programa debe haber detectado una plataforma conectada al PC, de este modo, se pueden seleccionar las bibliotecas que se deseen y el proyecto en el que serán incluidas las mismas.

El conjunto de las bibliotecas puede dividirse por tipo o por dispositivo. La división del software por tipo clasifica las bibliotecas en grupos dependiendo del nivel que ocupan dentro del programa de aplicación. Estos grupos son: Boards, Drivers, Components, Services, Applications y Utilities. La figura 25 muestra la clasificación por niveles de ASF.

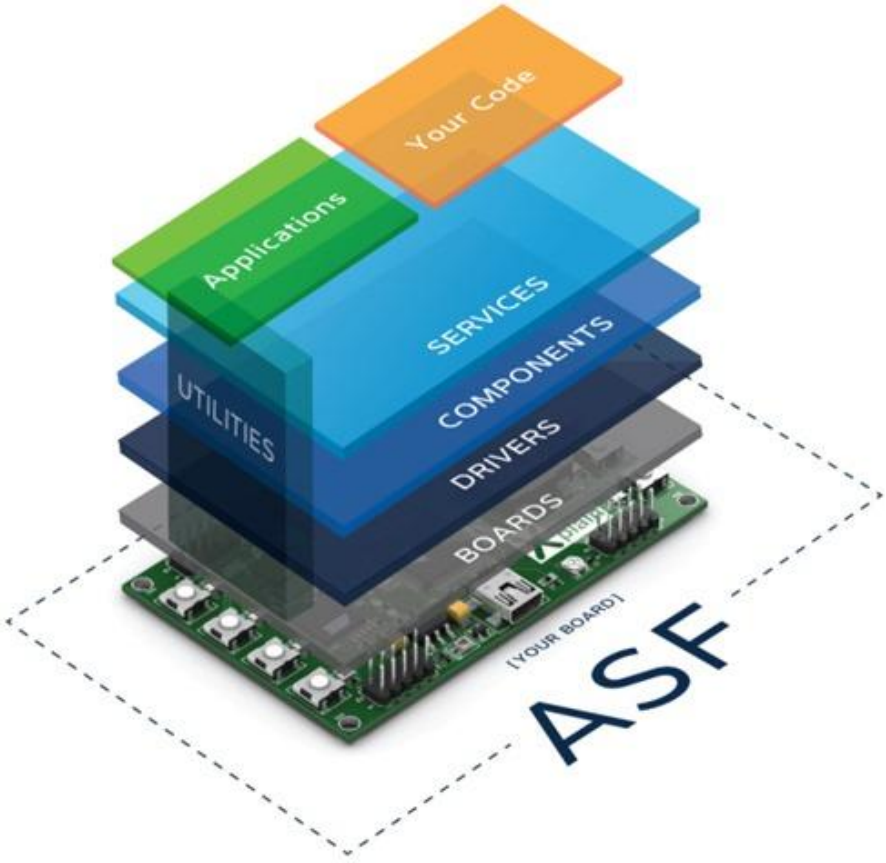

Figura 25. Clasificación por niveles de Atmel Software Framework.

También pueden clasificarse por dispositivo, mostrando las bibliotecas compatibles con un microcontrolador determinado.

7. HARDWARE

Para cumplir los objetivos 5 y 6 de este proyecto (referencia página objetivos) ha sido necesario diseñar una placa de circuito impreso que sea capaz de trazar caracteres en la pantalla LCD según lo que le indique el microprocesador (excitar los pines del LCD de segmentos con la tensión requerida en cada momento),. Esto es necesario porque el microcontrolador ATmega256RFR2 no dispone de un módulo hardware que se encargue de generar las señales adecuadas en función del tiempo que otros microcontroladores de Atmel si tienen incorporado. Es importante aclarar que no había disponible ninguna plataforma compatible con ZigBee que también lo fuese con el LCD por lo que en este proyecto se ha diseñado la placa de circuito impreso mencionada y los *drivers* software para poder controlar el LCD por la capa de aplicación.

Por otra parte, para alimentar los nodos mediante unas baterías, se ha diseñado un circuito de regulación para suministrar la corriente necesaria al microcontrolador y protegerlo de posibles picos de tensión. Este circuito ha sido implementado en una placa de prototipado y adaptado a los 3 nodos que forman la red.

En los siguientes epígrafes se encuentra documentado todo el desarrollo del proyecto relacionado con el hardware.

7.1. PLACA DE CIRCUITO IMPRESO PARA EL LCD

La placa de circuito impreso que ha sido fabricada tiene como principales funciones alimentar los pines del LCD con los niveles de tensión adecuados utilizando los pines de entrada/salida del microcontrolador y adaptar el LCD a la plataforma.

7.1.1. FUNCIONAMIENTO DEL LCD

Se trata de un LCD de tipo FSTN 96 segmentos controlables de manera individual. Setenta de los 96 segmentos están destinados a representar 5 caracteres alfanuméricos de 14 segmentos cada uno. La tensión de operación del LCD es Vlcd = 3.6 V y el método de control es 1/4 DUTY, 1/3 BIAS.

Que el método de control sea 1/4 DUTY significa que la trama de las señales que excitan el LCD tiene cuatro ciclos de trabajo, en los que la señal puede adoptar 4 niveles de tensión. El hecho de tener cuatro niveles de tensión diferentes determina el método de control a 1/3 BIAS.

Los segmentos se iluminan o no dependiendo de una combinación aritmética entre dos señales, una señal común a 24 segmentos denominada “COMx” donde x representa los números 0-3 dependiendo del segmento del que se trate y otra señal común a 4 segmentos denominada “SEGy” donde y representa los números 0-23 dependiendo del segmento del que se trate. Cada diferente combinación de COMx - SEGy está asociada a un segmento único, de forma que si existen 4 señales COM y 24 señales SEG, hacen un total de 96 combinaciones posibles, una cada para uno de los 96 segmentos.

Tanto las señales COM como las SEG son señales cuadradas con unos niveles de tensión específicos para la correcta iluminación de los segmentos. Las señales COM tienen los mismos niveles de tensión en cada ciclo de una trama, son señales comunes que no varían dependiendo del segmento que se desea iluminar. Cada señal COM está desfasada un ciclo con respecto a la anterior señal COM, de esta forma se consigue multiplexar en el tiempo 4 segmentos en cada señal SEG. La figura 26 muestra la forma de estas señales.

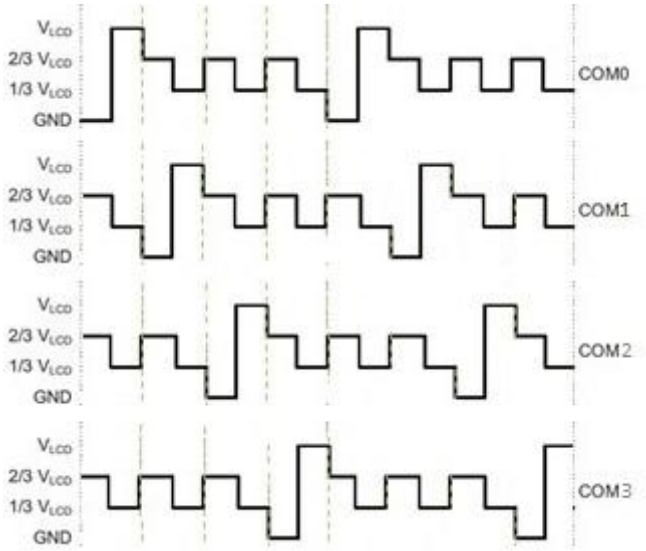

Figura 26. Forma de las señales COM0-3

Por otro lado, las señales SEG si que varían sus niveles de tensión en función de los segmentos que se deseen iluminar. Cada señal SEG puede controlar hasta 4 segmentos, uno segmento por cada ciclo de trabajo de la trama de forma que se multiplexan en el tiempo optimizando el número de pines necesarios para controlar el conjunto de segmentos. La figura 27 muestra la forma de las señales SEG donde se puede comprobar que los niveles de tensión están desfasados 1/2 ciclo con respecto a las señales COM para que los niveles de tensión no sean los mismos en el mismo instante de tiempo ya que de esta forma no se conseguirían los niveles de tensión deseados al restar ambas señales.

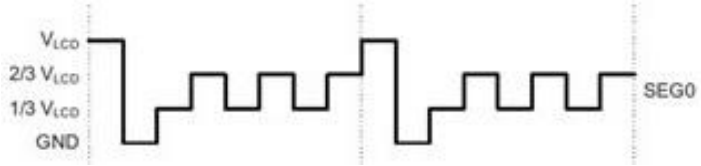

Figura 27. Forma de las señales SEG

Como se ha comentado en los párrafos anteriores, un segmento está asociado a una señal COMx y a una señal SEGy. Esto es así porque un segmento se ilumina si la diferencia de tensión de la señal SEGy-COMx en un ciclo de la trama es 2Vcld, por el contrario, si la diferencia de tensión es menor que el umbral 4/3Vlcd el segmento no se iluminará. La multiplexación en el tiempo de los segmentos dentro de una trama SEG y la sincronización con las señales COM dificulta el manejo del LCD. La figura 28 muestra un ejemplo de combinación de estas señales indicando que segmentos se iluminan y cuáles no. Los segmentos son representados como seg(0, 0) para el segmento asociado a SEG0 -COM0(primer ciclo de trabajo) y seg(0, 1) al segmento asociado a SEG0-COM1(segundo ciclo de trabajo).

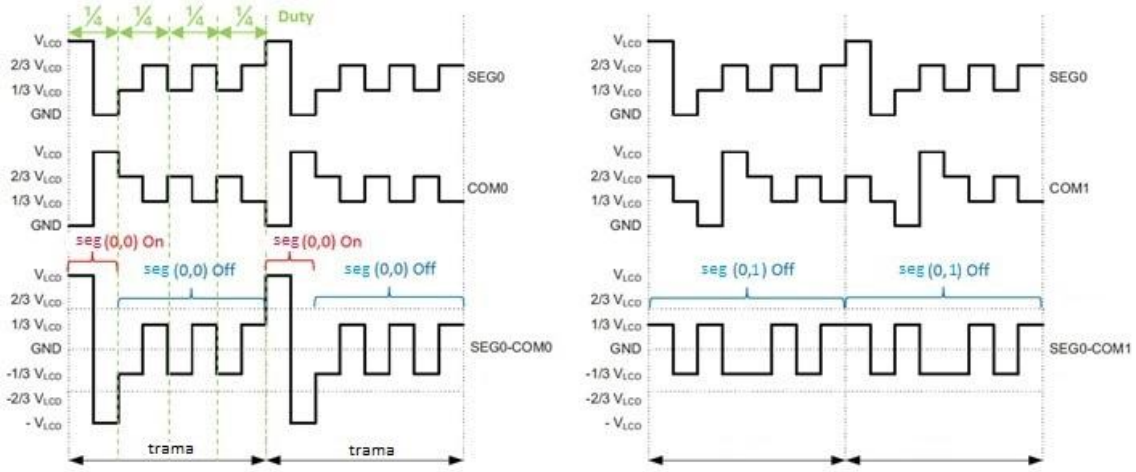

Figura 28. Ejemplo de combinación de las señales COM y SEG indicando que segmentos se iluminan.

7.1.2. DISEÑO DE LA PLACA DE CIRCUITO IMPRESO

Como se ha descrito en el apartado anterior, los niveles de tensión que deben excitar el LCD de segmentos son Vlcd = 3.6V, $\frac{2 \cdot \text{Vlcd}}{3}$, $\frac{\text{Vlcd}}{3}$ y 0V. Para conseguir los niveles de tensión se

van a utilizar los pines de entrada y salida de propósito general del microcontrolador junto con un divisor de tensión colocado entre el MCU y el LCD.

El circuito divisor de tensión para cada uno de las líneas se muestra en la figura 29.

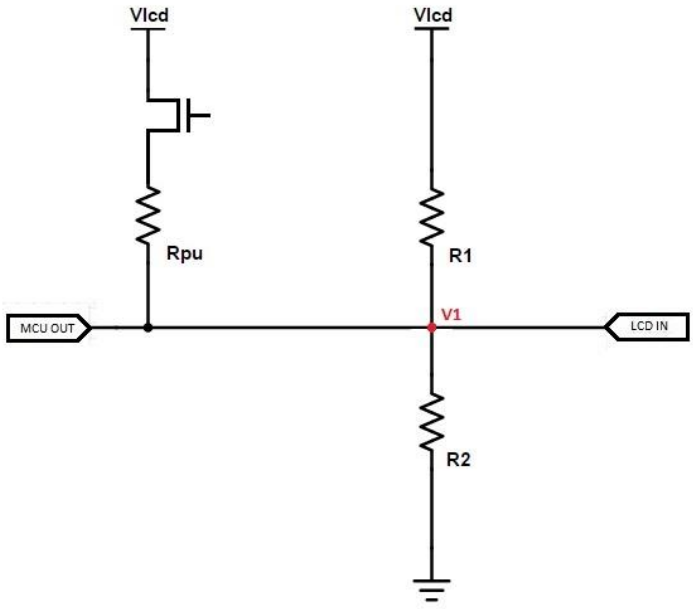

Figura 29. Circuito de conexión de cada uno de los pines del LCD con los pines del microcontrolador.

Como se puede ver en la figura x. cada uno de los pines del LCD está conectado con el microcontrolador a través de un divisor de tensión. Cuando el pin de entrada/salida correspondiente está configurado como entrada la impedancia de entrada al pin del MCU es muy elevada por lo que se puede considerar un abierto y la tensión V1 dependerá de si la resistencia de pull-up del microcontrolador está habilitada o no ya que modifica la resistencia equivalente al estar en paralelo con R1. Si el pin del microcontrolador está configurado como salida, la tensión en el LCD será la que fuerza el micro en la salida teniendo 0V o 3.6V dependiendo del valor del registro datos del puerto de entrada/salida correspondiente.

Para que los valores de V1 sean los deseados, las resistencias R1 y R2 deben tener un valor concreto en función de Rpu (resistencia de pull-up).

Los valores de tensión $\frac{2 \cdot \mathrm{Vlcd}}{3}$ y $\frac{\mathrm{Vlcd}}{3}$ se obtienen cuando el pin correspondiente del microcontrolador está configurado como entrada. Con esta configuración, la resistencia de pull-up puede estar habilitada o no dependiendo del valor del registro de control PORTXn. Para el caso en el que la resistencia de pull-up esta deshabilitada, se tiene:

$$\mathrm{I1} = \frac{\mathrm{Vlcd}}{(\mathrm{R1} + \mathrm{R2})}$$

$$V1 = \mathrm{I1} \cdot \mathrm{R2} = \frac{\mathrm{Vlcd}}{(\mathrm{R1} + \mathrm{R2})} \cdot \mathrm{R2} = \frac{\mathrm{Vlcd}}{3}$$

De donde se obtiene que:

$$\mathrm{R1} = 2 \cdot \mathrm{R2}$$

Por otro lado, para el caso en el que la resistencia de pull-up esta activa, se tiene que R1' es el paralelo de R1 y la resistencia Rpu:

$$R1' = \frac{R1 \cdot Rpu}{\mathrm{R1} + \mathrm{Rpu}}$$

El desarrollo es igual que en el caso anterior, con la única diferencia que en este caso el voltaje V1 deseado es $\frac{2 \cdot \mathrm{Vlcd}}{3}$. Entonces la ecuación para la tensión V1 es:

$$V1 = \mathrm{I1} \cdot \mathrm{R2} = \frac{\mathrm{Vlcd}}{(\mathrm{R1}' + \mathrm{R2})} \cdot \mathrm{R2} = \frac{2 \cdot \mathrm{Vlcd}}{3}$$

De donde se obtiene que:

$$2 \cdot R1' = R2$$

$$\frac{2 \cdot R1 \cdot Rpu}{\mathrm{R1} + \mathrm{Rpu}} = \mathrm{R2}$$

Sustituyendo por el resultado anterior obtenido R1 = 2·R2

$$\frac{2 \cdot 2 \cdot R2 \cdot Rpu}{2 \cdot \mathrm{R2} + \mathrm{Rpu}} = \mathrm{R2}$$

$$4 \cdot R2 \cdot Rpu = 2 \cdot R2^2 + \mathrm{R2} \cdot \mathrm{Rpu}$$

$$4 \cdot Rpu = 2 \cdot \mathrm{R2} + \mathrm{Rpu}$$

$$R2 = 3 \cdot \frac{Rpu}{2}$$

Una vez obtenido este resultado, directamente se obtiene el valor de R1:

$$\mathrm{R1} = 2 \cdot \mathrm{R2} = 3 \cdot \mathrm{Rpu}$$

En el datasheet del microcontrolador se especifica que el valor de las resistencias de pull-up para los pines de entrada/salida tiene un valor de 78.7 KΩ. Por lo tanto, el valor de las resistencias R1 y R2 para conseguir los niveles de tensión requeridos es el siguiente:

R1 = 236 KΩ

R2 = 115 KΩ

Antes de realizar el diseño de la placa de circuito impreso, deben realizarse unas pruebas para comprobar si realmente las señales se generan de forma correcta. Estas pruebas son realizadas con un osciloscopio con ayuda de una protoboard una vez que el software para controlar los puertos de entrada/salida está terminado. De esta forma se puede comprobar si tanto el hardware como el software funcionan correctamente antes de diseñar la PCB. Las figuras 30, 31, 32 y 33. muestran unas capturas en el osciloscopio de las señales generadas durante la prueba con los cursores situados en distintas posiciones. En la figura w. se puede comprobar que la tensión de pico es de 3.44V = Vlcd, en la figura x. que el segundo nivel de tensión es de 2.20V, en la figura y. que el tercer nivel de tensión es de 1.12V, valores muy aproximados a $\frac{2 \cdot Vlcd}{3}$ y $\frac{Vlcd}{3}$ respectivamente. Por otro lado, en la figura z. se puede ver que la frecuencia de la trama es de 61.7Hz, tal y como se esperaba.

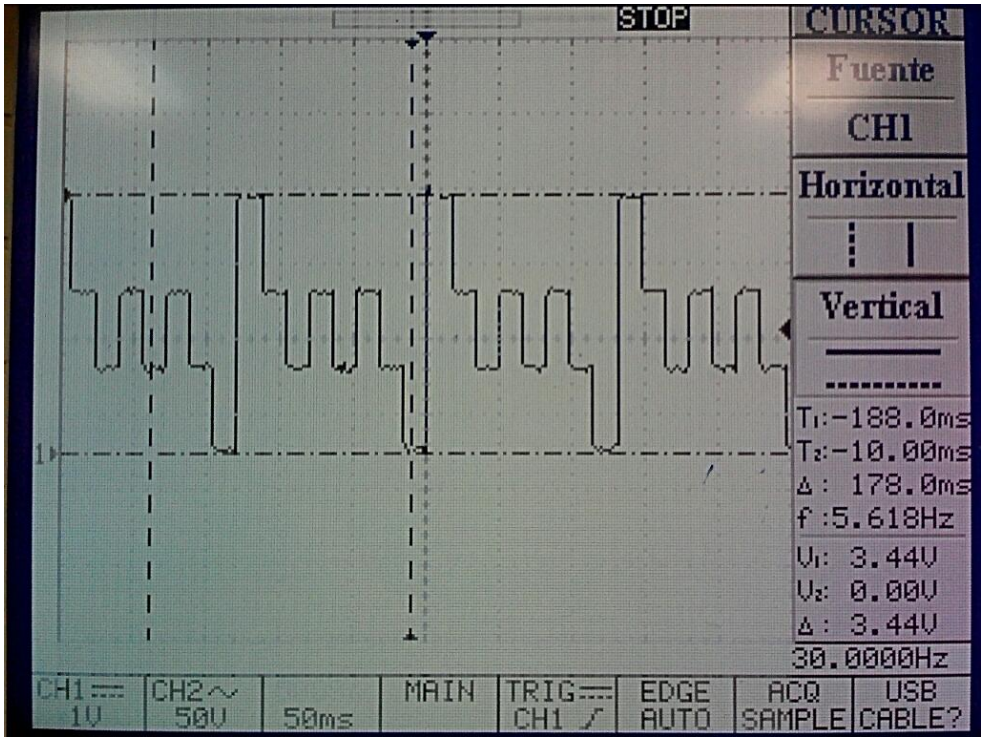

Figura 30. Captura de la señal generada que muestra el valor de tensión de pico de la señal.

Figura 31. Captura de la señal generada que muestra el valor del segundo nivel de tensión.

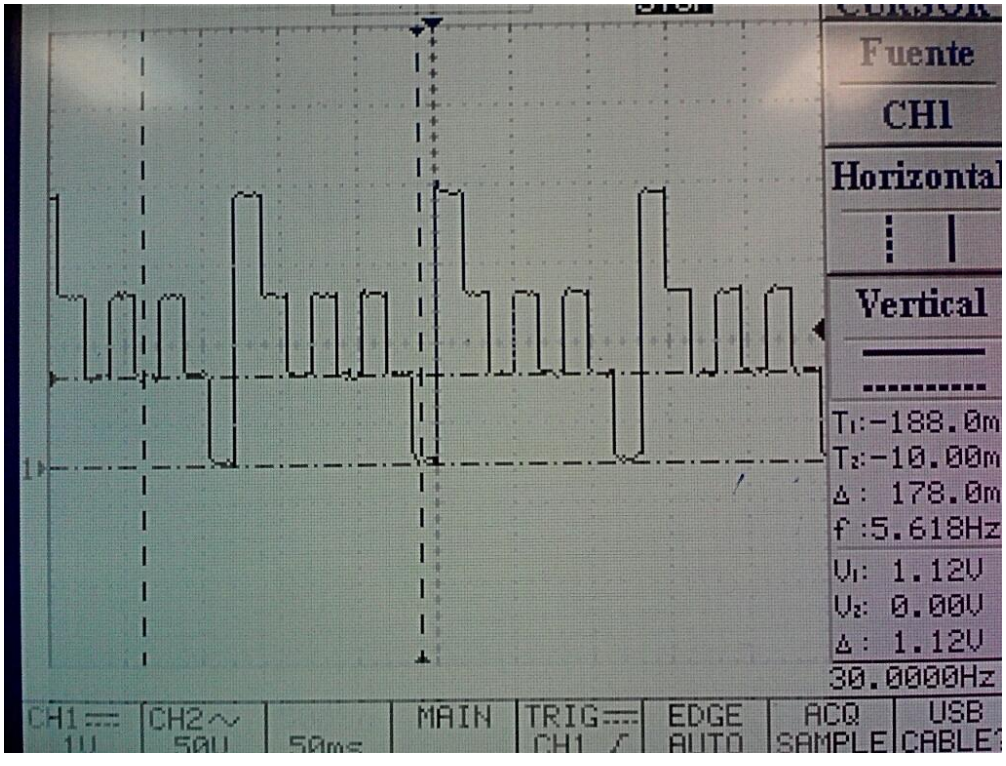

Figura 32. Captura de la señal generada que muestra el valor del tercer nivel de tensión.

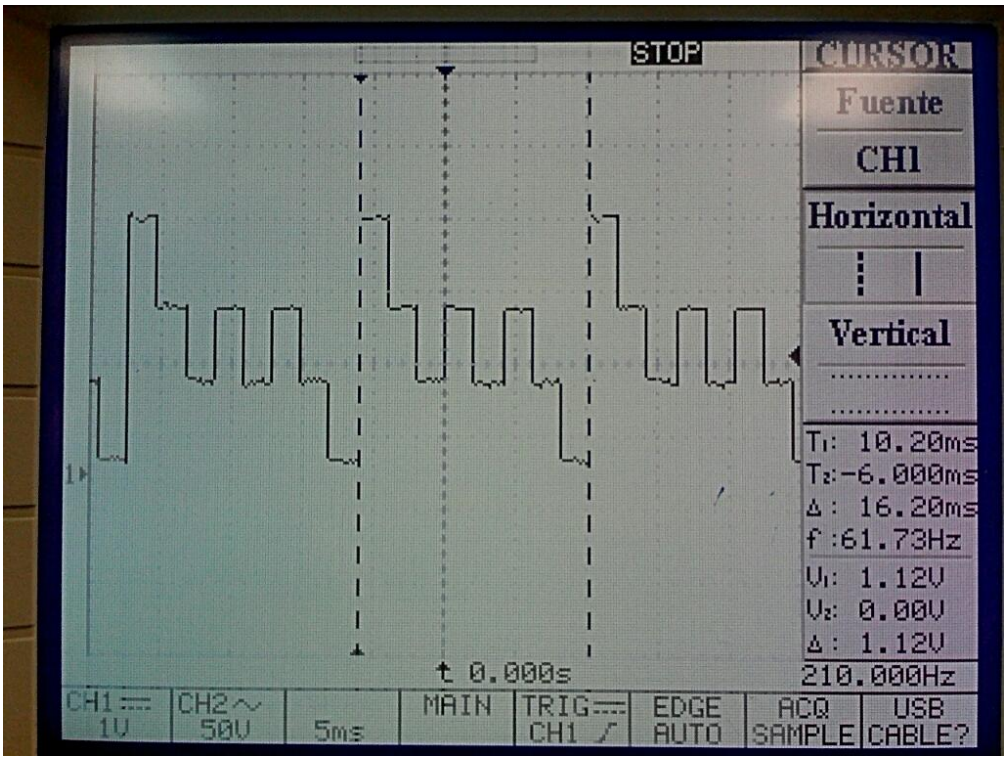

Figura 33. Captura de la señal generada que muestra la frecuencia de la trama.

Para realizar el diseño de la placa de circuito impreso se ha utilizado el programa Orcad Capture para el esquemático y Orcad Layout para diseñar la PCB a partir del esquemático realizado. El circuito es una red de resistencias basadas en el circuito que se muestra en la "Figura x. circuito de uno de los pines del LCD", teniendo un divisor de tensión para línea que conecta un puerto de entrada/salida del microcontrolador con un pin del LCD.

La tabla 5 muestra una lista de los componentes utilizados para la fabricación de la placa de circuito impreso junto con los conectores.

COMPONENTE	CANTIDAD	DESCRIPCIÓN
Hirose DF9-51P-1V(69)	1	Conector macho recto de 51 pines, 2 filas, paso 1mm, terminación en SMD (Surface Mounting Device).
Preci-Dip 499-87-220-10-003101	3	Conector hembra. Ángulo de 90°, 20 pines, 2 filas, paso 2.54 mm, montaje en orificio pasante.
PANASONIC ERJP08F1202V	19	Resistencia, 12K, 1% de tolerancia
PANASONIC ERJP08F3602V	19	Resistencia, 36K , 1% de tolerancia
BOURNS CR1206-FX-1003ELF	19	Resistencia, 100K, 1% de tolerancia

PANASONIC ERJ8GEYJ204V	19	Resistencia, 200K, 5% de tolerancia

Tabla 5. Componentes de fabricación de la placa de circuito impreso.

El footprint de las resistencias utilizado en el diseño de la PCB es el 1206, para el conector Hirose DF9-51P-1V(69) ha sido necesario crear un nuevo footprint debido a que no existía ninguno con el mismo número de pines y las mismas dimensiones.

Utilizando el Layout de Orcad se ha realizado el diseño de una PCB de dos capas separando en cada capa las resistencias que están conectadas a VCC y las que lo están a GND para conseguir un tamaño más reducido. El mapeo de los puertos en el software se ha ordenado de tal forma que no haya cruces de pistas en la placa de circuito impreso. Las figuras 34, 35 y 36 muestran el layout por separado de las capas y en conjunto.

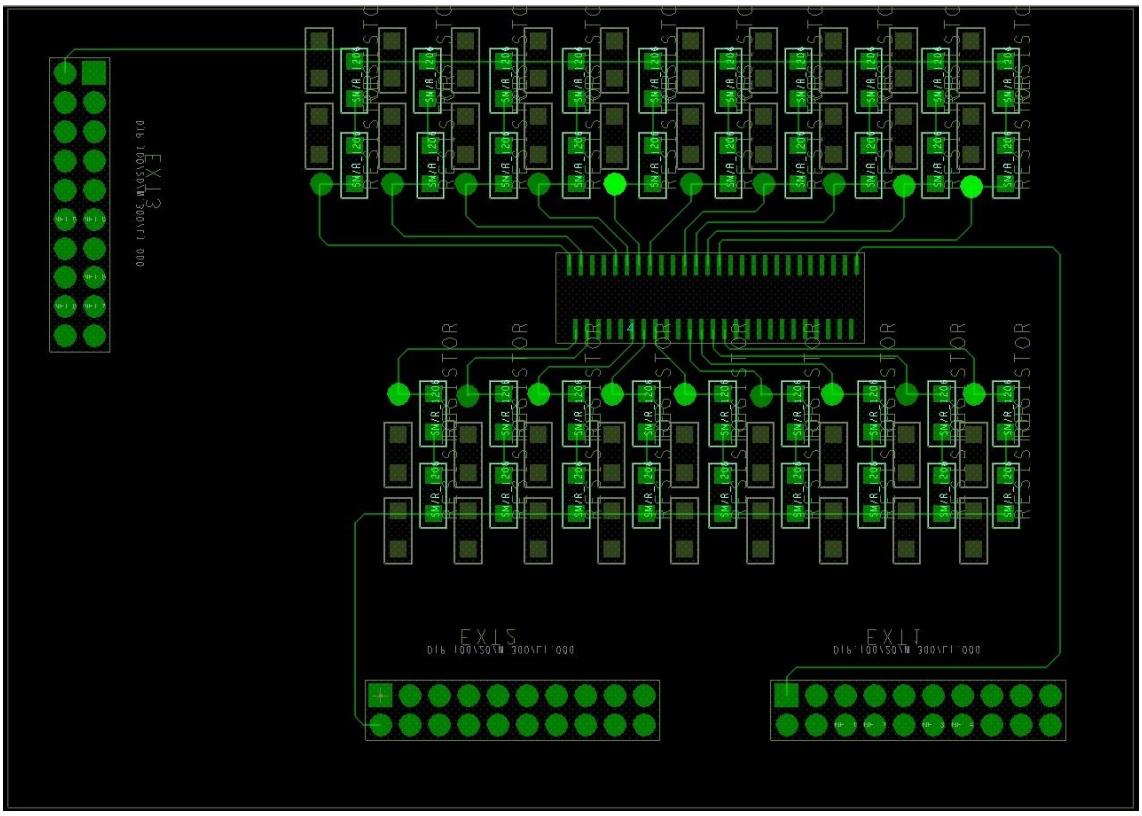

Figura 34. Layout de la capa superior de la PCB.

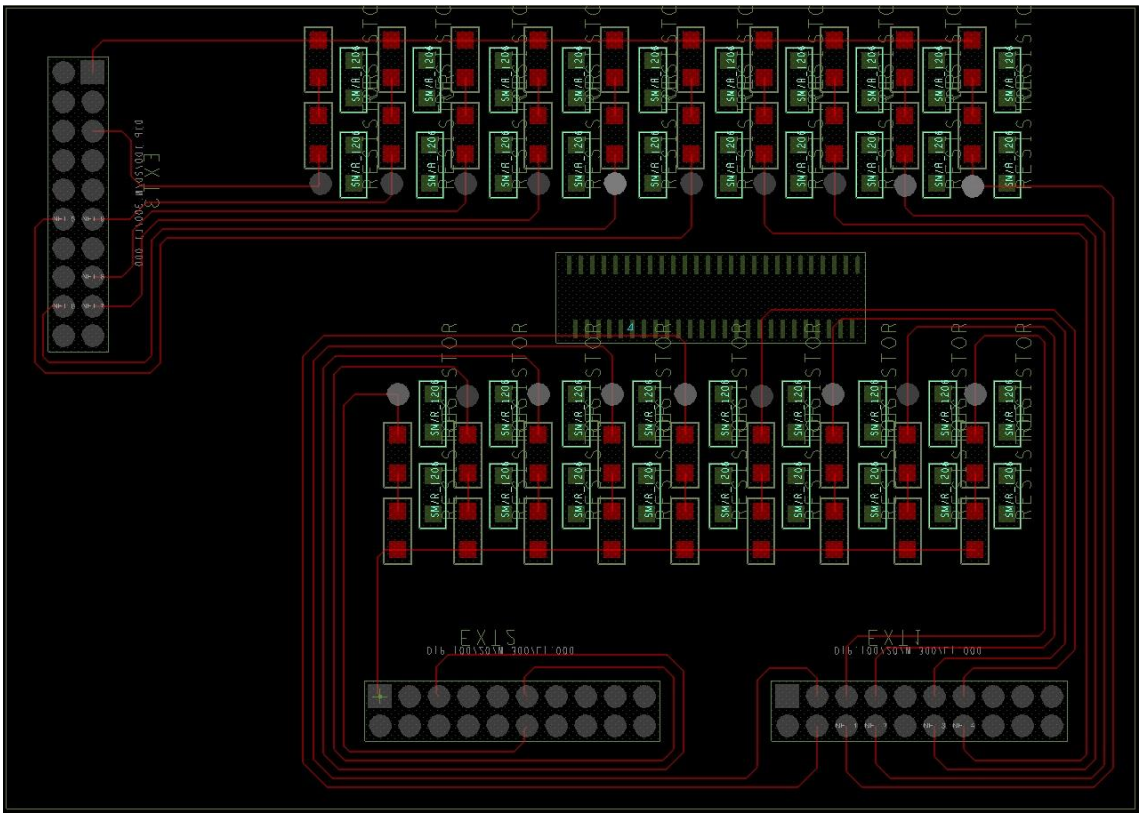

Figura 35. Layout de la capa inferior de la PCB.

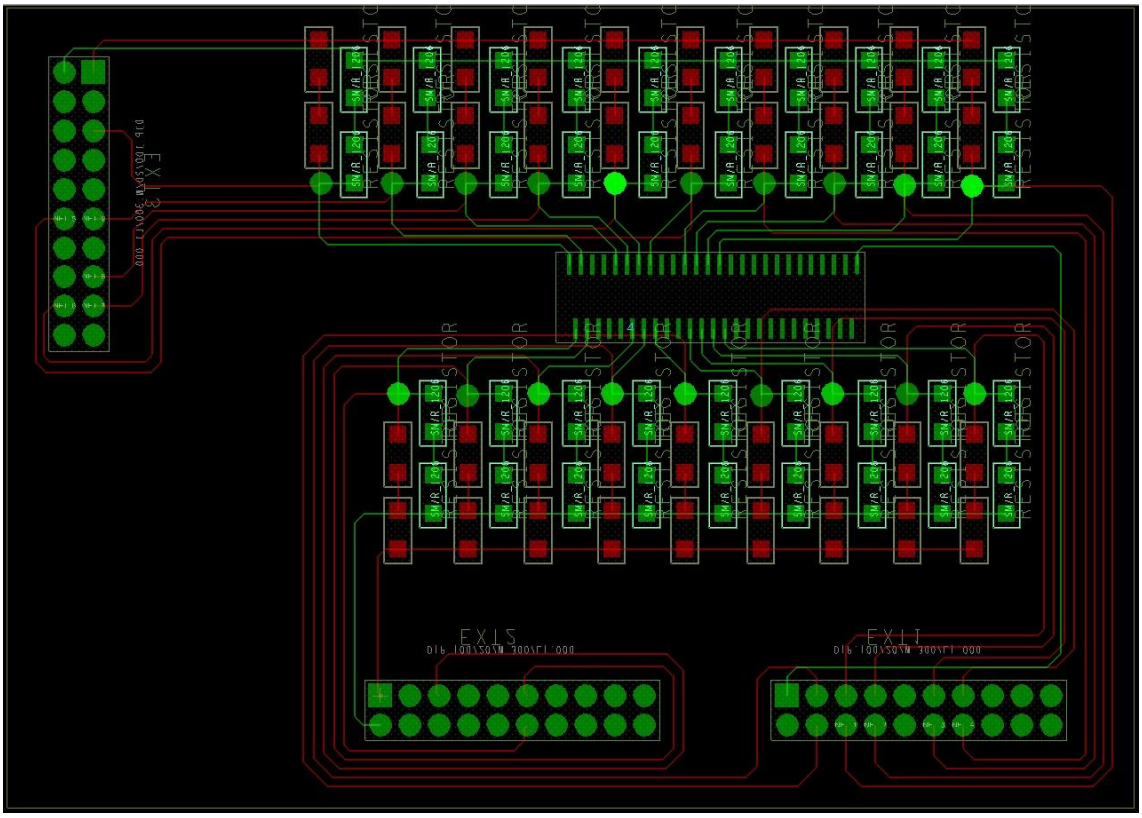

Figura 36. Layout de las dos capas de la PCB.

Una vez fabricada la placa de circuito impreso se soldaron las resistencias, el conector para el LCD y los conectores para el conexionado con la plataforma ATmega256RFR2 Xplained Pro. La figura 37 muestra el resultado final de la PCB conectada a la plataforma.

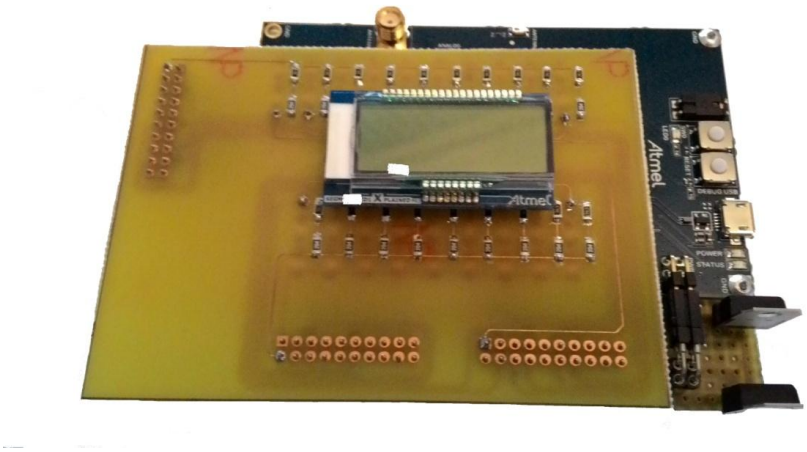

Figura 37. Placa de circuito impreso de adaptación del LCD conectada a la plataforma.

7.2. ALIMENTACIÓN

La plataforma Atmega256RFR2 Xplained Pro debe ser alimentada con voltajes de 3.3 y 5V con una corriente mínima y máxima de 500mA y 2.5A respectivamente. El valor mínimo de corriente de alimentación se corresponde con el mínimo de corriente necesaria para garantizar el correcto funcionamiento de la plataforma, aún conectando el máximo número de placas de extensión y el valor máximo es el límite de corriente que no debe superarse para garantizar la seguridad de la plataforma.

La etapa de alimentación ha sido implementada en una placa de prototipado, utilizando los reguladores MC7805 y LM317 y una pila como fuente de energía.

7.2.1. REGULADOR MC7805 DE 5V

El regulador MC7805 es un regulador lineal que va a ser utilizado para conseguir una tensión de alimentación de 5V y una corriente máxima de 2A.

Para conseguir que la corriente de salida del regulador sea superior a los 500mA necesarios para que el microcontrolador pueda funcionar correctamente junto todos los periféricos que sean conectados, la diferencia de tensión entre la entrada y la salida debe

tener un valor mínimo. En la figura 38 se puede comprobar que para una temperatura de 25ºC, se obtiene una corriente de salida de 500mA para Vin-Vout = 1.75V. Por lo tanto, si se desea tener una corriente superior a esta, la tensión proporcionada por la batería deberá ser mayor que Vout + 1.75V = 6.75V.

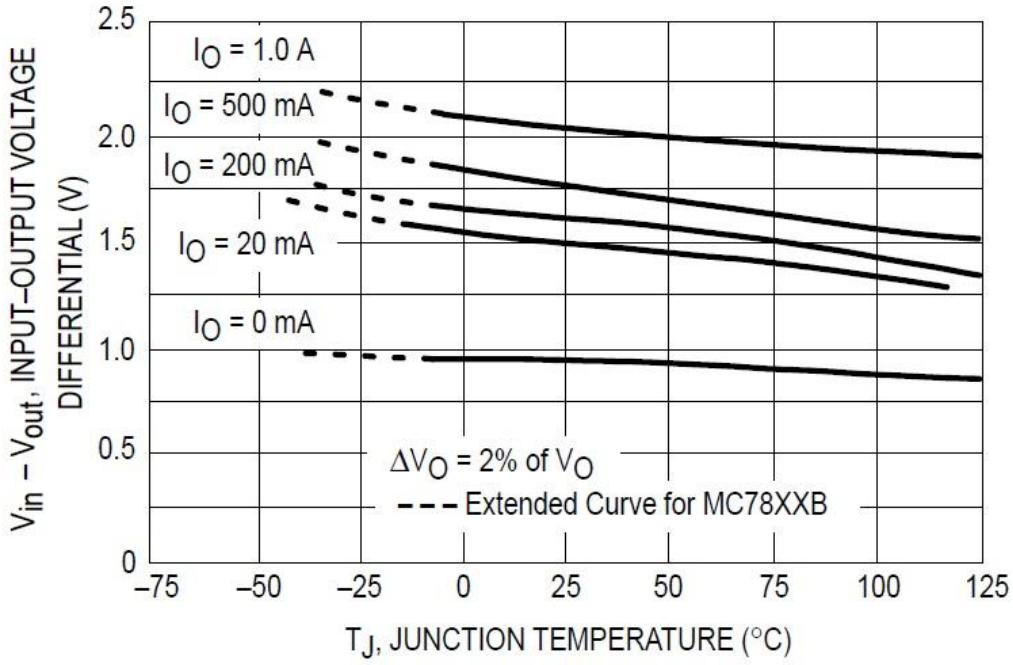

Figura 38. Diferencia de tensión mínima para que el regulador sea capaz de proporcionar una corriente de salida de 500mA.

Dado que la batería que va a ser utilizada tiene una tensión en bornes de 9V, la diferencia de tensión entre la entrada y la salida es de 4V. La figura 39 muestra la relación entre esta diferencia de tensión y la corriente de salida del regulador, teniendo Io = 2A. Este valor de corriente no supera el límite de corriente de seguridad del microcontrolador, que es de 2.5A.

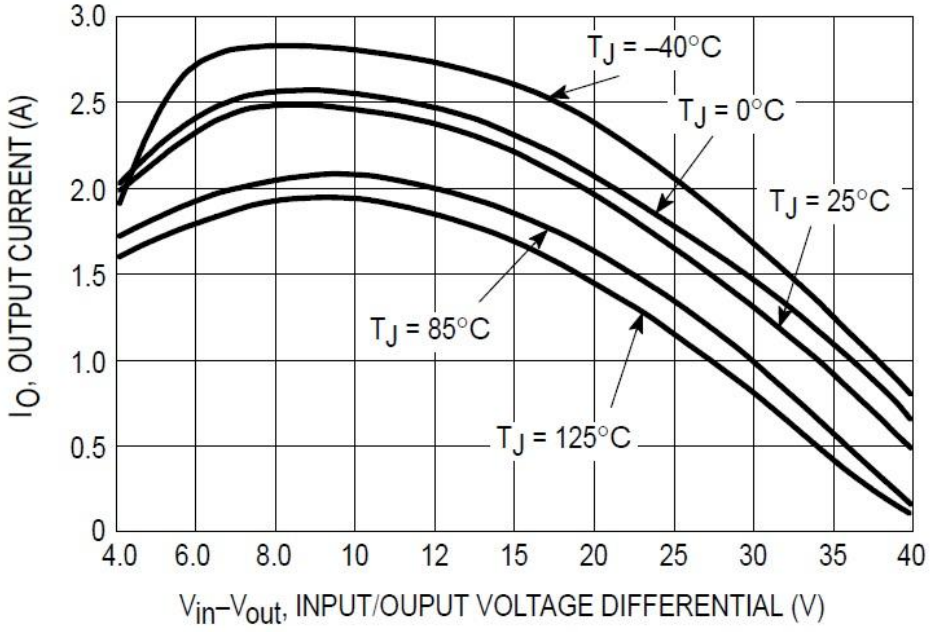

Figura 39. Corriente de salida del regulador en función de la diferencia de tensión Vin-Vout y la temperatura ambiente.

El consumo de potencia del regulador depende de la corriente de salida del mismo, de la forma: $Pd = (\mathrm{Vin} - \mathrm{Vo}) \cdot \mathrm{Iout}$, por lo que dependerá de la corriente que consuma el microcontrolador.

7.2.2. REGULADOR LM317

El regulador LM317 es un regulador de tensión lineal con tensión de salida ajustable. La tensión a la salida del regulador que se desea obtener es de 3.3V, necesaria para suministrar energía a los periféricos del microcontrolador junto con una corriente de salida mayor a 500mA. La curva de corriente en función de la diferencia de tensión Vin-Vout tiene una pendiente 1A/V hasta llegar a los 5V de diferencia, por lo que si se ajusta demasiado esta diferencia de tensión, la más mínima variación podría hacer que la corriente de salida fuese menor a 500mA. Para una batería de 9V, la corriente de salida tendría un máximo de 2.25A que no supera el límite de 2.5A como se puede comprobar en la figura 40 muestra la relación entre la diferencia de tensión de entrada y salida y la corriente de salida máxima del regulador.

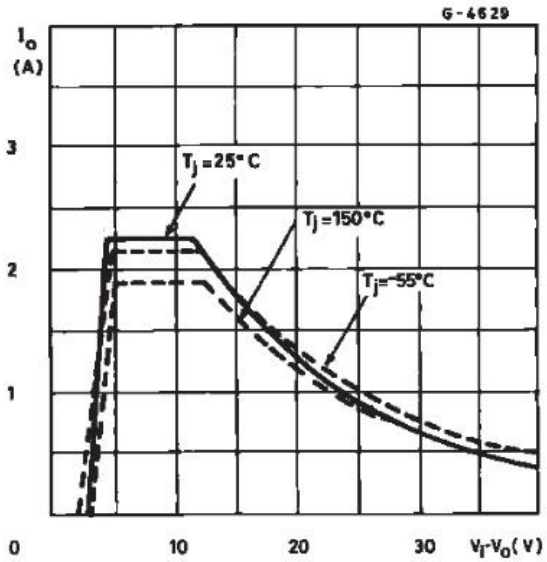

Figura 40. Relación entre la diferencia de tensión de entrada y salida del regulador y la corriente de salida máxima.

La tensión de salida se ajusta variando el valor de dos resistencias dada una tensión de referencia entre los pines 1 y 3 del regulador. Los valores de las resistencias, teniendo en cuenta que la tensión de referencia es de 1.25V son calculados a continuación. La figura 41 muestra el circuito que ajusta la tensión de salida del regulador.

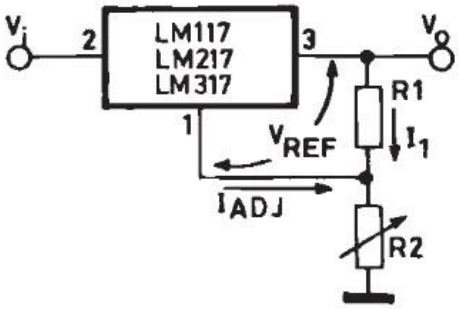

Figura 41. Circuito para ajustar la tensión de salida del regulador LM317.

Para una tensión de referencia Vref=1.25V, R1=4.7KΩ y una corriente de ajuste Iadj=100uA, se tiene que:

$$Vo = \mathrm{Vref} \cdot \left(1 + \frac{\mathrm{R2}}{\mathrm{R1}}\right) + \mathrm{Iadj} \cdot \mathrm{R2}$$

$$R2 = \left(\frac{\mathrm{Vo} - \mathrm{Vref}}{\frac{\mathrm{Vref}}{\mathrm{R1}} + \mathrm{Iadj}} \right) = 5.6\ \mathrm{K\Omega}$$

El consumo de potencia del regulador depende de la corriente de salida del mismo, de la forma: $Pd = (\mathrm{Vin} - \mathrm{Vo}) \cdot \mathrm{Iout}$, por lo que dependerá de la corriente que consuma el microcontrolador.

7.2.3. ESQUEMÁTICO

Los pines de los reguladores están asignados según las figuras 42 y 43.

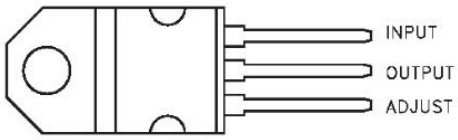

Figura 42. Pinout del regulador LM317.

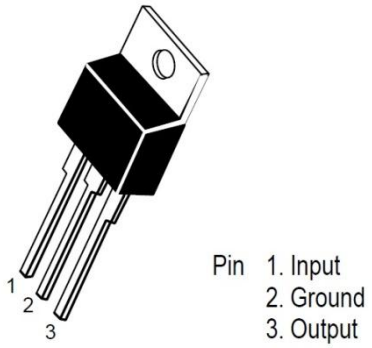

Figura 43. Pinout del regulador MC 7805.

El circuito de alimentación de las plataformas está formado por ambos reguladores. Este circuito ha sido implementado en una placa de prototipado y se ha conectado a los tres nodos de la red. La figura 44 muestra el esquemático del circuito, mientras que la figura 45 muestra una captura del circuito implementado conectado a la plataforma.

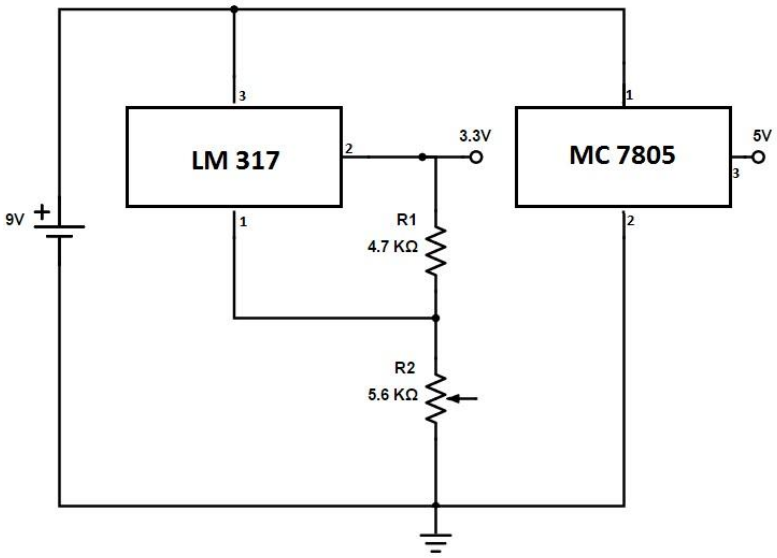

Figura 44. Esquemático del circuito de regulación.

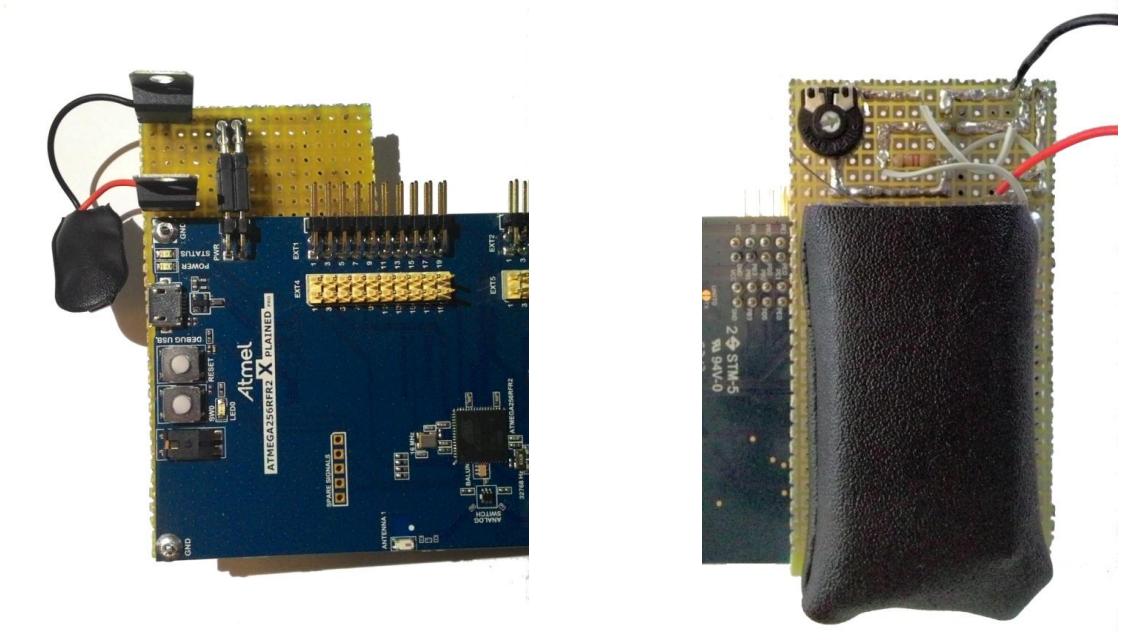

Figura 45. Captura del circuito de regulación implementado en la placa de prototipado.

8. SOFTWARE

Para la realización de este proyecto han sido desarrollados los drivers para controlar el LCD de segmentos, varias bibliotecas de utilidad relacionadas con el procesado de los datos de temperatura, la interfaz de usuario, la detección de pulsaciones del botón de software de corta y larga duración, el sensor de temperatura y la comunicación mediante el protocolo TWI (2-Wires Interface), además de los programas de aplicación del coordinador y los dispositivos finales donde se implementa la comunicación mediante el protocolo ZigBee.

8.1. STACK ZIGBEE

El stack ZigBee implementa las capas Física y MAC del protocolo proporcionando todos los servicios requeridos por el estándar IEEE 802.15.4. La arquitectura del mismo se muestra en la figura 46.

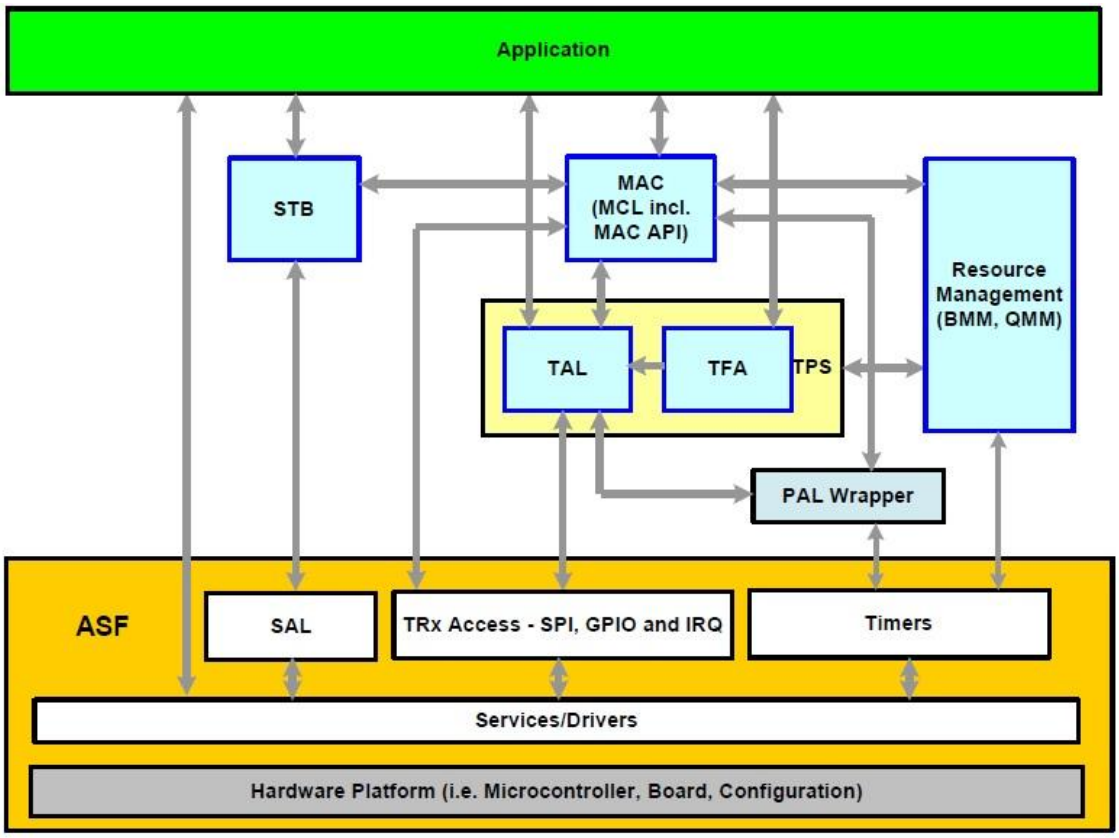

Figura 46. arquitectura del stack ZigBee

La MAC API es la interfaz principal entre la capa de aplicación y el stack. Se utiliza para enviar solicitudes y mensajes de respuesta al núcleo de la capa MAC, "MCL" en la imagen, así como para recibir información a través de funciones de callback implementadas en la aplicación. El TAL (Transceiver Abstraction Layer) contiene la funcionalidad del transceptor especificada por el estándar IEEE 802.15.4 controlando el envío y la recepción de tramas incluyendo reenvío de las mismas y envío automático de ACKs, el control de acceso al medio mediante CSMA, la detección de energía del canal y la gestión de potencia. Existe una interfaz TAL API que también puede ser utilizada por la aplicación.

Para que el stack funcione correctamente es necesario llamar a la función mac_task() de forma frecuente desde la aplicación de forma que se actualicen las máquinas de estados de MCL, TAL y PAL y se procesen sus colas de eventos.

8.2. SOFTWARE DEL COORDINADOR

El programa del nodo coordinador está dividido en dos partes, la primera de ellas controla la comunicación ZigBee y es ejecutada en cada iteración de un bucle infinito situado en el main a través de la función mac_task(). La otra parte es la aplicación, que también se ejecuta en cada iteración de este bucle a través de la función app_task() para actualizar el LCD. Aparte de esta función, la aplicación comprende todas las funciones de callback para comunicarse con la capa MAC. El procesado de los datos de temperatura se realiza cada vez que se reciben dichos datos, de forma que solo se ejecuta cuando es necesario. Por otro lado, tanto la detección de pulsaciones como el control de la máquina de estados de la interfaz de usuario son controlados mediante una interrupción que se ejecuta cada 20ms.

La figura 47 muestra las dependencias del programa de aplicación.

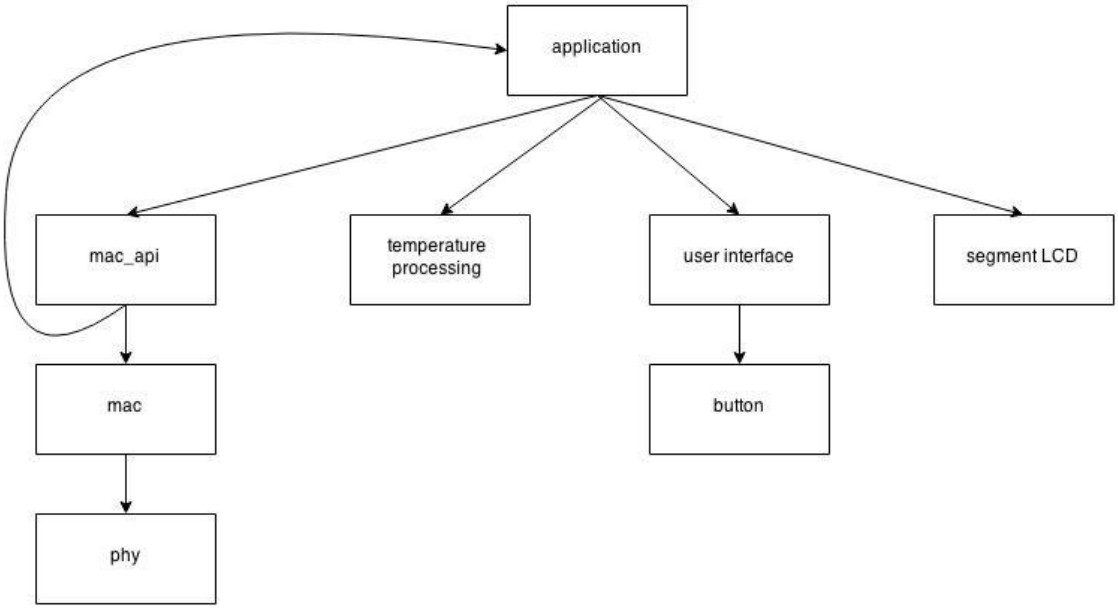

Figura 47. dependencias del programa de aplicación del nodo coordinador.

8.2.1. COMUNICACIÓN

El programa de aplicación del coordinador implementa dos tipos de funciones para crear la red y establecer comunicación con el resto de nodos. Entre ellas están las funciones que solicitan un servicio de la capa MAC y las funciones de callback que son llamadas desde la capa MAC cuyo cuerpo está definido en la aplicación.

El procedimiento de asociación, iniciado por un dispositivo final, consiste en la indicación de la solicitud de asociación a la capa de aplicación a través de la función de callback usr_mlme_associate_ind(). En esta función se evalúa la posibilidad de que se asocie un nodo a la red, dependiendo de la capacidad de la misma y el número de nodos que se encuentran asociados en ese momento y se responde al dispositivo final a través de la función wpan_mlme_associate_resp() indicando la dirección corta (16 bits) del dispositivo. Las direcciones son asignadas de forma que todos los nodos tengan direcciones diferentes. La asociación de un nuevo dispositivo a la red se indica mediante el parpadeo del led naranja durante 4 segundos.

El procedimiento de desasociación puede ser iniciado tanto por el coordinador como por el dispositivo final. Si es iniciado por el coordinador, la solicitud se envía a través de la función wpan_mlme_disassociate_req() indicando la dirección del nodo que se quiere eliminar de la red. Si por el contrario la solicitud es iniciada por un dispositivo final, se le notifica a la capa de aplicación mediante la función de callback usr_mlme_disassociate_ind().

La transmisión de datos se realiza de forma directa al coordinador con origen en los nodos finales. Cuando el coordinador recibe una trama de datos, la aplicación tiene constancia de ello

a través de la función de callback usr_mcps_data_ind() que contiene como parámetros la longitud del mensaje, la carga útil, la calidad del canal estimada durante la recepción de la trama y la dirección corta de origen de la trama. Cada vez que se recibe una trama se utilizan los parámetros de entrada de esta función para procesar los 2 Bytes de temperatura recibidos en la carga útil, la dirección de origen para conocer a que nodo debe asignarse esa temperatura y la estimación de la calidad del canal para poder visualizar los parámetros de la red en el LCD.

El código referente a la comunicación se ha implementado en los ficheros app.h y app.c que pueden encontrarse en el ANEXO 15.1.

8.2.2. DRIVERS PARA EL SLCD

El LCD debe ser controlado mediante una serie de señales cuadradas con cuatro niveles de tensión diferentes. Para generar estos niveles de tensión se utiliza el divisor de tensión junto con los pines de entrada y salida del microcontrolador como ha sido descrito en el apartado 7.1.2.

La trama de control es cíclica, tiene una frecuencia de 60Hz y está dividida en cuatro ciclos de trabajo. Las tramas se refieren a la forma de onda de las señales SEG y COM por lo que para iluminar una serie de segmentos determinados en el LCD, la forma de onda de estas señales será la misma en cada trama.

Dentro de esta trama, un segmento solo se ilumina en un ciclo si la señal SEG que lo controla tiene un pulso de anchura Vlcd-GND en el ciclo de trabajo correspondiente, produciendo una intermitencia en la iluminación del segmento imperceptible para el ojo humano a esta frecuencia. La tabla 6 muestra la correspondencia de los segmentos con las señales SEG y COM. Las cuatro columnas representan una trama y una columna, un ciclo.

	COM0	COM1	COM2	COM3	Comments
SEG0	G1	G2	G4	G3	Atmel logo, 4 stage battery-, Dot-point-, usb- and play indicator
SEG1	G0	G6	G7	G5	
SEG2	E7	E5	E3	E1	4 stage wireless-, AM-, PM- Volt- and milli volt indicator
SEG3	E6	E4	E2	E0	
SEG4	A0-h	A0-i	A0-k	A0-n	1st 14-segment character.
SEG5	B3	A0-f	A0-e	A0-d	
SEG6	A0-a	A0-b	A0-c	B4	
SEG7	A0-g	A0-j	A0-l	A0-m	
SEG8	A1-h	A1-i	A1-k	A1-n	2nd 14-segment character
SEG9	B2	A1-f	A1-e	A1-d	
SEG10	A1-a	A1-b	A1-c	B5	
SEG11	A1-g	A1-j	A1-l	A1-m	
SEG12	A2-h	A2-i	A2-k	A2-n	3rd 14-segment character
SEG13	B1	A2-f	A2-e	A2-d	
SEG14	A2-a	A2-b	A2-c	B6	
SEG15	A2-g	A2-j	A2-l	A2-m	
SEG16	A3-h	A3-i	A3-k	A3-n	4th 14-segment character.
SEG17	B0	A3-f	A3-e	A3-d	
SEG18	A3-a	A3-b	A3-c	B7	
SEG19	A3-g	A3-j	A3-l	A3-m	
SEG20	A4-h	A4-i	A4-k	A4-n	5th 14-segment character. Celsius and Fahrenheit indicator.
SEG21	B8	A4-f	A4-e	A4-d	
SEG22	A4-a	A4-b	A4-c	B9	
SEG23	A4-g	A4-j	A4-l	A4-m	

Tabla 6. Correspondencia de los segmentos con las señales SEG y COM

Por ejemplo, si nos fijamos en la tabla, el primer ciclo de la señal SEG4 controla el segmento A0-h, o lo que es lo mismo, el segmento “h” del carácter 1. La figura 48 muestra la distribución de los segmentos en el LCD.

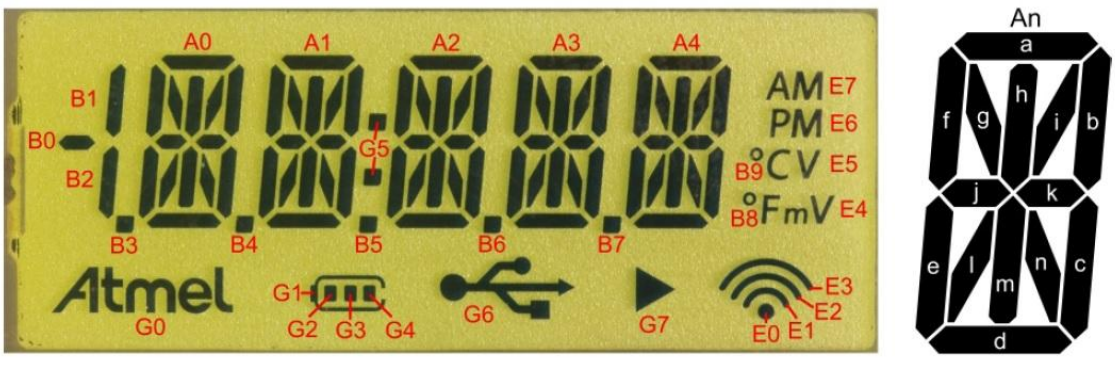

Figura 48. Distribución de los segmentos en el LCD.

Para conseguir generar los niveles de tensión correctos en cada uno de los pines y en cada ciclo se ha diseñado una lógica mediante software que, a partir de unos patrones de bits asignados para cada carácter alfanumérico que se desee representar en el LCD se obtengan unos valores para mapear los registros de los puertos de entrada/salida del microcontrolador con el valor correspondiente en cada uno de los ciclos. La figura 49 muestra el diagrama de flujo del programa para obtener los valores de mapeo de los registros.

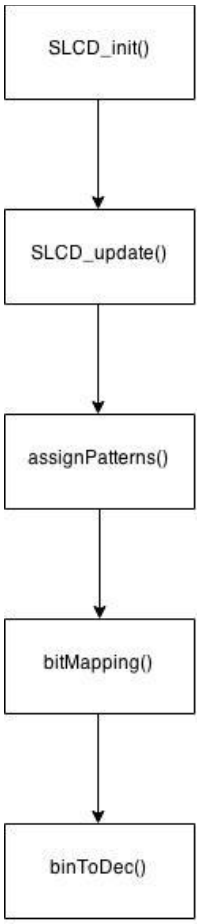

Figura 49. Diagrama de flujo del programa para obtener los valores de mapeo de los registros.

SLCD_init: Inicializa todos los puertos como entradas y la fase de la trama a 0. Habilita las interrupciones globales y el timer/counter0 como fuente de interrupción, realiza un preescalado del clk del periférico a clk/8 para que la trama tenga una frecuencia de aproximadamente 60Hz e inicia el contador de pulsos. Cuando llega a 255 pulsos (8 bits) se genera la interrupción y comienza a contar de nuevo desde 0.

SLCD_update: tiene como argumentos 4 enteros que indican los caracteres alfanuméricos que se desean representar en el LCD. es la función que será utilizada por la aplicación para actualizar el LCD con los caracteres que se pasen como parámetros. La relación entre el valor de los parámetros y el carácter alfanumérico asociado que se representará en el LCD es la siguiente:

VALOR DEL PARÁMETRO	CARÁCTER ALFANUMÉRICO REPRESENTADO
< 0 >	< 0 >
< 1 >	< 1 >
< 2 >	< 2 >
< 3 >	< 3 >
< 4 >	< 4 >
< 5 >	< 5 >
< 6 >	< 6 >
< 7 >	< 7 >
< 8 >	< 8 >
< 9 >	< 9 >
< 10 >	< 0ºC >
< 11 >	< 1ºC >
< 12 >	< 2ºC >
< 13 >	< 3ºC >
< 14 >	< 4ºC >
< 15 >	< 5ºC >
< 16 >	< 6ºC >
< 17 >	< 7ºC >
< 18 >	< 8ºC >
< 19 >	< 9ºC >
< 20 >	< esp >
< 21 >	< E >
< 22 >	< R >
< 23 >	< A >
< 24 >	< P >
< 25 >	< C >
< 26 >	< T >

De esta forma, si llamamos a la función SLCD_update(1, 2, 3, 4) el carácter 1 representado en el LCD será un uno, el segundo un 2, el cuarto un 3 y el quinto un 4. El carácter 3 del LCD se utiliza como espacio debido que no había suficientes pines de entrada/salida del microcontrolador para controlar todos los caracteres por lo que se decidió tomar esta decisión para poder mostrar un indicador a la izquierda del LCD junto con su valor a la derecha del mismo.

assingPatterns: Cada patrón está compuesto por cuatro filas y cuatro columnas, las filas representan las cuatro señales SEG que controlan un carácter y las cuatro columnas, los cuatro ciclos de trabajo. Representa con un <0> los segmentos que no deben iluminarse y con un <1> los segmentos que deben iluminarse para representar un carácter alfanumérico concreto. Existe una relación entre el índice de la tabla constante <patterns> y el caracter que se desea representar. Cada 4 elementos del array están relacionados con el patrón de un carácter determinado, de esta forma los 4 primeros elementos contienen el patrón del carácter <0>, los 4 siguientes elementos contienen el patrón del carácter <1> y así sucesivamente. Utilizando

esta relación se asignan los patrones correspondientes al valor de los parámetros de SLCD_update.

bitMapping: esta función se encarga de realizar un mapeo en los puertos PB, PE, PD, PF y PG del microcontrolador en función del valor de los patrones asignados para cada ciclo de trabajo. La tabla 7 puede ayudar a comprender como se realiza este mapeo.

	CICLO DE TRABAJO				
SEÑAL	1	2	3	4	
	0	0	0	0	
SEG11	x	x	x	x	--> patterns.pattern_2 [3]
SEG22	x	x	x	x	--> patterns.pattern_4 [2]
SEG9	x	x	x	x	--> patterns.pattern_2 [1]
	0	0	0		
SEG17	x	x	x	x	--> patterns.pattern_3 [1]
SEG16	x	x	x	x	--> patterns.pattern_3 [0]
SEG4	x	x	x	x	--> patterns.pattern_1 [0]
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
SEG7	x	x	x	x	--> patterns.pattern_1 [3]
SEG18	x	x	x	x	
COM2	0	0	1	0	
SEG10	x	x	x	x	
SEG21	x	x	x	x	
SEG20	x	x	x	x	
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
SEG19	x	x	x	x	
COM0	1	0	0	0	
COM1	0	1	0	0	
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
SEG6	x	x	x	x	
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
SEG23	x	x	x	x	
	0	0	0	0	
COM3	0	0	0	1	
	0	0	0	0	
	0	0	0	0	
SEG8	x	x	x	x	
	0	0	0	0	
	0	0	0	0	

Tabla 7. Valores de mapeo de los puertos PB, PE, PD, PF y PG del microcontrolador

Los patrones son distribuidos por la tabla de mapeo de forma que los pines de salida queden ordenados de tal forma que no haya cruces de pistas en la placa de circuito impreso fabricada. La forma de mapear los puertos es la siguiente: se ha dividido la tabla en los colores rojo, azul, verde, naranja y negro. De esta forma, los bits de las primeras ocho filas marcadas en rojo serán asignados al PB del microcontrolador, asignando los valores de cada columna en un ciclo diferente. El resto de bloques de 8 filas x 4 columnas se asignan respectivamente al PE, PD, PF y PG desde arriba hacia debajo de la tabla.

binToDec: Una vez realizada la asignación de valores de los puertos para cada ciclo, estos son almacenados en una serie de estructuras que después serán utilizadas en la interrupción para modificar el valor de los registros de forma adecuada.

La figura 50 muestra el diagrama de flujo de la rutina de interrupción.

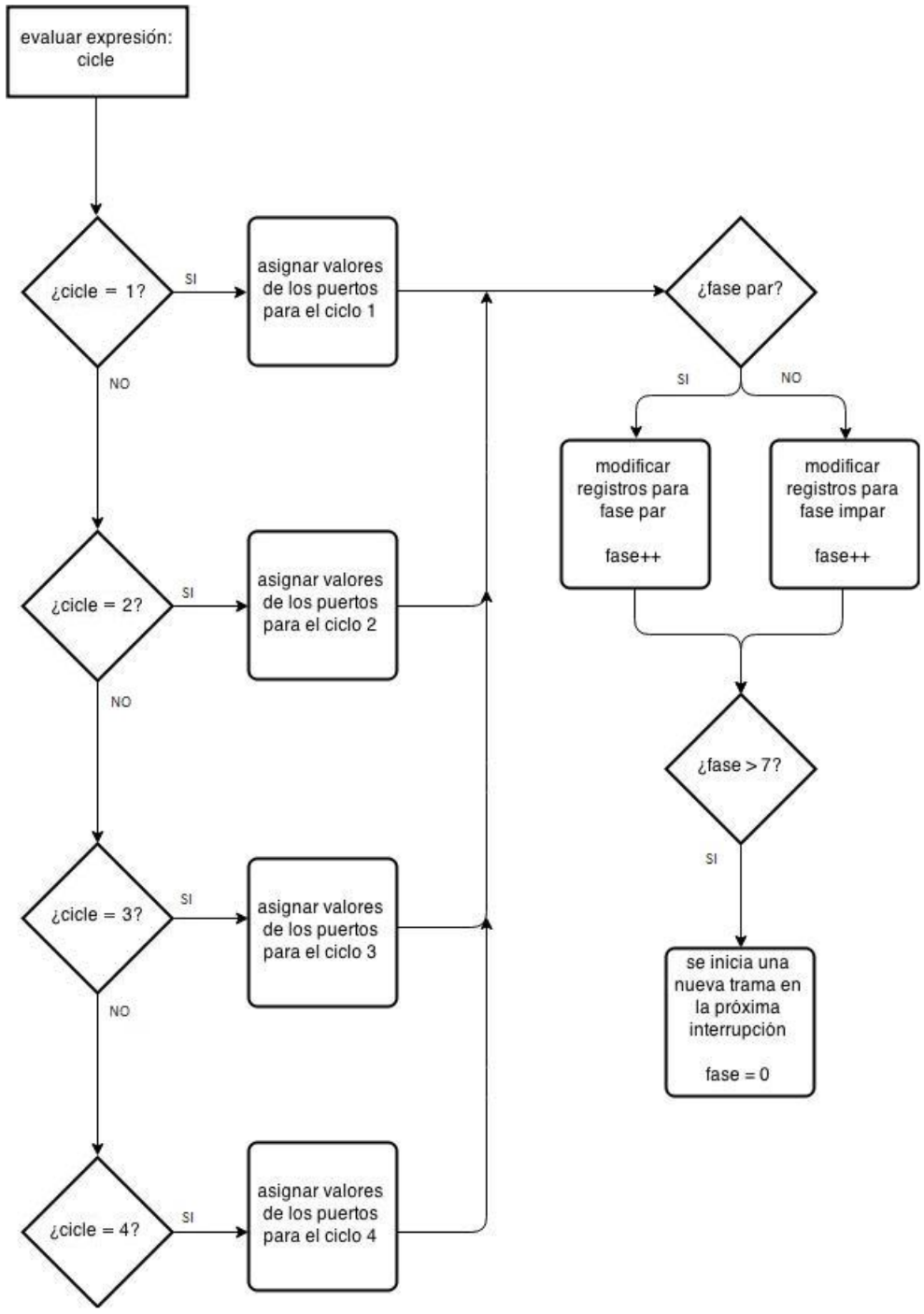

Figura 50. diagrama de flujo de la rutina de interrupción del LCD.

La rutina de interrupción se ejecuta para cada fase de la trama, es decir, dos veces por cada ciclo de trabajo. Se utiliza esta frecuencia debido a que en cada ciclo de trabajo las señales deben generar una señal cuadrada con dos niveles de tensión, siendo esta rutina la que se encarga de modificar los registros de los puertos de entrada/salida del microcontrolador. Los valores para mapear los puertos guardados en unas estructuras son utilizados en la rutina de interrupción, pero estos valores no modifican directamente los registros, sino que se utiliza una relación lógica entre el estado de los segmentos (0 = no

iluminado, 1 = iluminado) junto con la fase del ciclo, par o impar, en la que nos encontremos y el valor de los registros. Las figuras x. e y. pueden ayudar a comprender estas relaciones. La tabla 8 muestra el valor de tensión que debe obtenerse para generar las señales correctamente en función del estado del segmento, la fase y el tipo de señal (SEG o COM). Por otro lado, la tabla 9 muestra el valor de los puertos PORTxn y DDRxn, donde x es el puerto y b el pin del mismo, en función del valor de tensión deseado a la salida.

VALOR	1		0	
FASE	IMPAR	PAR	IMPAR	PAR
SEG	0	V	2V/3	V/3
COM	V	0	V/3	2V/3

Tabla 8. valor de tensión que debe obtenerse a la salida para generar las señales correctamente en función del estado del segmento, la fase y el tipo de señal (SEG o COM).

PORTxn	DDRxn	VOLTAJE
0	1	0
1	1	V
0	0	V/3
1	0	2V/3

Tabla 9. valor de los puertos PORTxn y DDRx, en función del valor de tensión deseado a la salida.

El código de los drivers del LCD se ha implementado en los ficheros SLCD_Driver.h y SLCD_Driver.c que pueden encontrarse en el ANEXO 15.1.

8.2.3. PROCESADO DE LOS DATOS DE TEMPERATURA

La figura 51 muestra el diagrama de flujo del programa implementado para procesar los datos de temperatura recibidos por el coordinador.

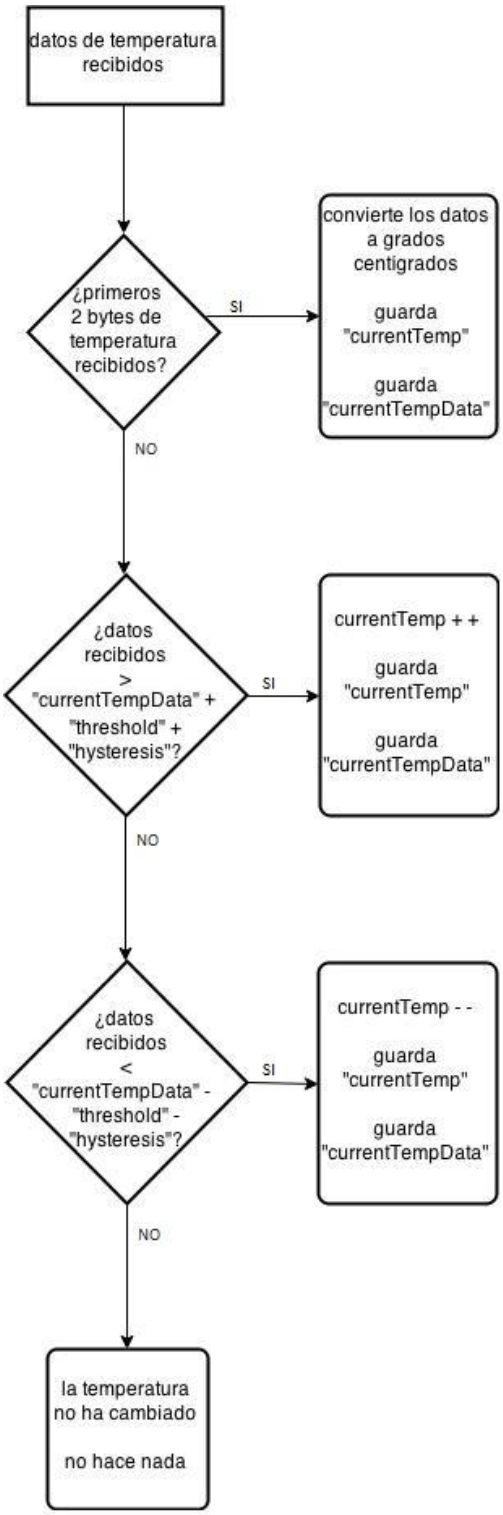

Figura 51. diagrama de flujo del procesado de los datos de temperatura.

Los datos de temperatura enviados por los dispositivos finales ocupan 2 bytes. La primera vez que envían estos datos se realiza una conversión a grados centígrados. Para ello, primero se deben ordenar correctamente los 2 bytes en un uint16_t, después se utiliza el valor

absoluto multiplicado por 0.0625. Este valor lo proporciona el datasheet del sensor de temperatura para convertir a grados centígrados los datos con una resolución de 12 bits.

Los siguientes datos de temperatura recibidos por parte de un nodo, pueden aumentar o disminuir en un grado el valor de temperatura monitorizado dependiendo de si sobrepasa unos umbrales relativos aplicados al valor de temperatura en el que se encontraba el nodo. Si este umbral de cambio de temperatura es fijo, puede ocurrir que la señal de la temperatura sensada oscile en este umbral, lo que produciría que el valor de temperatura monitorizado cambia constantemente. Para evitar esto se ha incluido un margen de histéresis al umbral, de forma que si el nuevo valor de temperatura es mayor al valor actual + el umbral + el margen de histéresis se incrementa la temperatura y si es menor al valor actual – el umbral – el margen de histéresis se decrementa la temperatura consiguiendo un intervalo entre cambios de temperatura en vez de un umbral fijo. Para aumentar la precisión en los cambios del valor de temperatura se utiliza el valor CurrentTempData que almacena los datos no transformados a grados centígrados para trabajar con una resolución de 0.0625 ºC.

La figura 52 muestra los cambios del valor de temperatura mostrado en función de la señal de temperatura. Como se puede comprobar en la imagen, si no se utilizase histéresis y el umbral de cambio fuese fijo se tendrían seis cambios del valor de temperatura en ese intervalo de tiempo y teniendo en cuenta la histéresis solo cambia de valor en dos ocasiones.

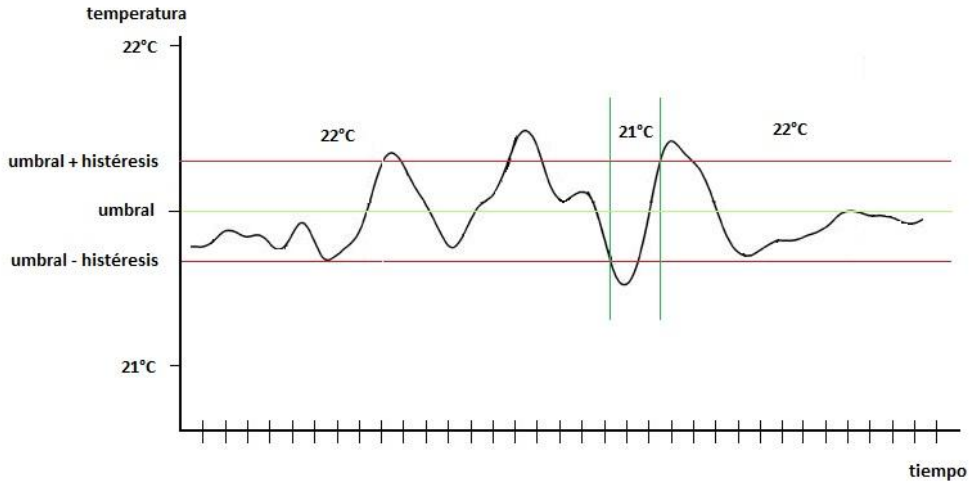

Figura 52. Valores de temperatura en función de la señal teniendo en cuenta márgenes de histéresis.

El código referente al procesado de temperatura se ha implementado en los ficheros temp.h y temp.c que pueden encontrarse en el ANEXO 15.1.

8.2.4. INTERFAZ DE USUARIO

La interfaz de usuario gestiona los datos que se muestran en el LCD en función de los eventos de pulsación del botón entrantes. Se basa en una máquina de estados que se actualiza cada 20ms mediante una función de interrupción y que llama a la función buttonUpdate() que gestiona los eventos de pulsaciones para conocer si hay un evento de pulsación. Gracias la diferenciación entre pulsación larga y pulsación corta es posible controlar una interfaz de usuario de forma sencilla e intuitiva con un solo botón, moviéndonos por un menú principal mediante pulsaciones cortas y entrando en el submenú de visualización del parámetro que se desee mediante una pulsación larga. Una vez dentro de un submenú, se podrá ver la información de todos los nodos de la red mediante pulsaciones cortas y volverá al menú principal con una pulsación larga del botón.

CONTROL

Esta biblioteca tiene la función de detectar eventos de pulsación, siendo capaz de distinguir entre pulsaciones largas y pulsaciones cortas. Este evento es utilizado por la biblioteca de interfaz de usuario para cambiar de estado.

La figura 53 muestra el diagrama de estados de la detección de pulsaciones.

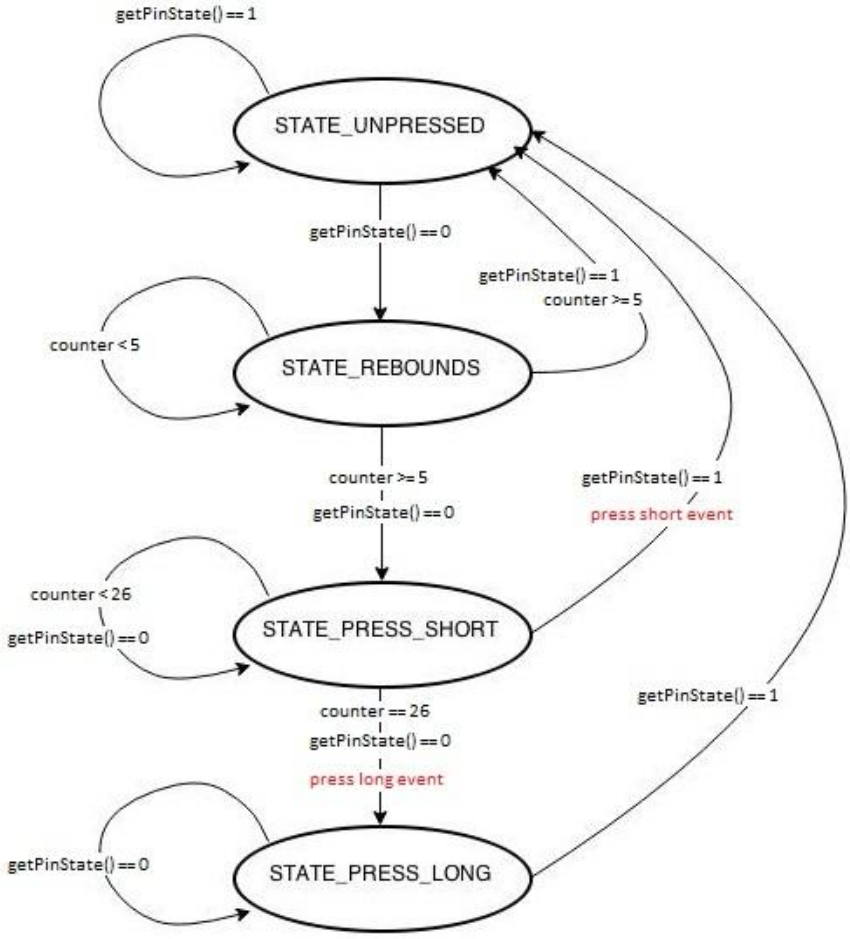

Figura 53. Diagrama de estados de la detección de pulsaciones del botón.

La función getPinState() es una función estática de la biblioteca “button” que devuelve el estado del pin 4 del puerto B del microcontrolador: PINB4 conectado al botón que debe estar configurado como entrada. La pulsación del botón produce un cortocircuito del pin de salida a GND por lo que el estado lógico del pin será 0 y getPinSatate() devolverá el valor 0. Por lo tanto, cuando el botón no está pulsado el nivel lógico es 1 ya que el valor de tensión del pin es +VCC por lo que getPinState() devolverá 1. El circuito de conexión del botón al pin de entrada del MCU se encuentra en el esquemático del microcontrolador en el ANEXO 15.3.

El algoritmo es una máquina de estados que detecta el tiempo de pulsación utilizando un contador. El algoritmo se ejecuta cada 20ms, incrementando el contador si el botón sigue pulsado.

El estado inicial es STATE_UNPRESSED y permanecerá así mientras no se detecte ninguna pulsación. Si se detecta una pulsación mediante getPinSatate() == 0, se pasa al estado STATE_REBOUNDS en el que permanecerá durante 100ms sin tener en cuenta el estado del pin hasta que pasen los rebotes (counter > 5). Una vez pasado el tiempo de rebotes, se vuelve a

comprobar el estado del pin: si está pulsado, entonces se podrá contabilizar como pulsación corta cambiando el estado a STATE_SHORT_PRESS, pero el evento de pulsación corta no se envía hasta que se suelte el botón ya que puede tratarse de una pulsación larga. Cuando el botón se mantiene pulsado el tiempo suficiente como para que counter>26, quiere decir que el botón ha permanecido pulsado 520ms por lo que el estado cambia a STATE_LONG_PRESS. Para determinar los tiempos de pulsación corta y larga se han realizado pruebas hasta ajustarlo perfectamente. De esta forma se tiene que:

- Tiempo de pulsación < 100ms, no hay pulsación
- 100ms < Tiempo de pulsación < 520ms, pulsación corta
- Tiempo de pulsación > 520ms, pulsación larga.

El código referente a la detección de pulsación del botón se ha implementado en los ficheros button.h y button.c que pueden encontrarse en el ANEXO 15.1.

REPRESENTACIÓN DE LA INFORMACIÓN

La interfaz de usuario se divide en un menú principal y en submenús en los que se representa la información de los nodos relativa al parámetro del menú principal seleccionado. El LCD se actualiza en cada iteración del programa de aplicación cuyos valores representados dependen del estado de la interfaz de usuario. El método de control de la información que se desea visualizar se realiza a través de un botón, de forma que la máquina de estados se actualiza dependiendo del evento devuelto por la función buttonUpdate(). De forma genérica, una pulsación corta se utiliza para movernos dentro de un menú o un submenú y una pulsación larga para entrar a un submenú o salir de él.

La información que puede visualizarse en el menú es la siguiente:

- **TR:** Tamaño de la Red. Indica el número de nodos que forman la red ZigBee.
- **CE:** Calidad del Enlace. Indica mediante un valor de porcentaje 0-100% la calidad del enlace con cada nodo de la red. Para ello se utiliza el parámetro LinkQuality que hace una estimación de la EB/No mientras se recibe cada trama. Este parámetro puede servir de utilidad a la hora de situar los nodos en la etapa de integración de la red, evitando localizaciones perjudicadas por el multitrayecto.
- **PT:** Paquetes Transmitidos. Muestra el orden de los paquetes transmitidos al coordinador por cada nodo de la red. Permite comprobar el flujo de datos de la red en cada uno de los enlaces y verificar el correcto funcionamiento de los nodos.

- **TE:** TEmperatura. Permite monitorizar la temperatura en cada uno de los nodos de la red.

La figura 54 muestra el diagrama de estados de la interfaz de usuario donde short press representa la entrada de un evento de pulsación corta y long press representa la entrada de un evento de pulsación larga.

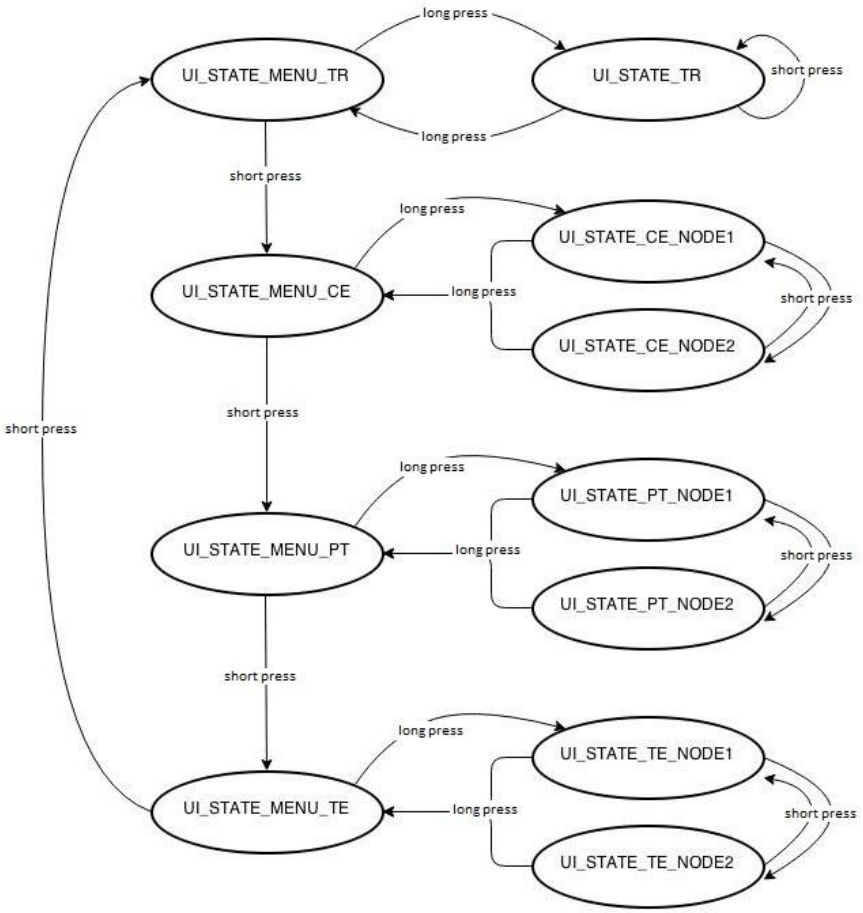

Figura 54. Diagrama de estados de la interfaz de usuario.

El código referente a la interfaz de usuario se ha implementado en los ficheros ui.h y ui.c que pueden encontrarse en el ANEXO 15.1.

8.3. SOFTWARE DEL DISPOSITIVO FINAL

El programa de los nodos finales está dividido en dos partes al igual que el programa del coordinador, la primera de ellas controla la comunicación ZigBee y es ejecutada en cada iteración de un bucle infinito situado en el main a través de la función mac_task(). La otra parte es la aplicación, que también se ejecuta en cada iteración de este bucle a través de la función app_task() para establecer comunicación con el coordinador enviando los datos de temperatura cuando el nodo este despierto, comunicarse con el sensor para obtener los datos de temperatura y dormir el nodo. Aparte de esta función, la aplicación comprende todas las funciones de callback para comunicarse con la capa MAC.

La figura 55 muestra las dependencias del programa de aplicación.

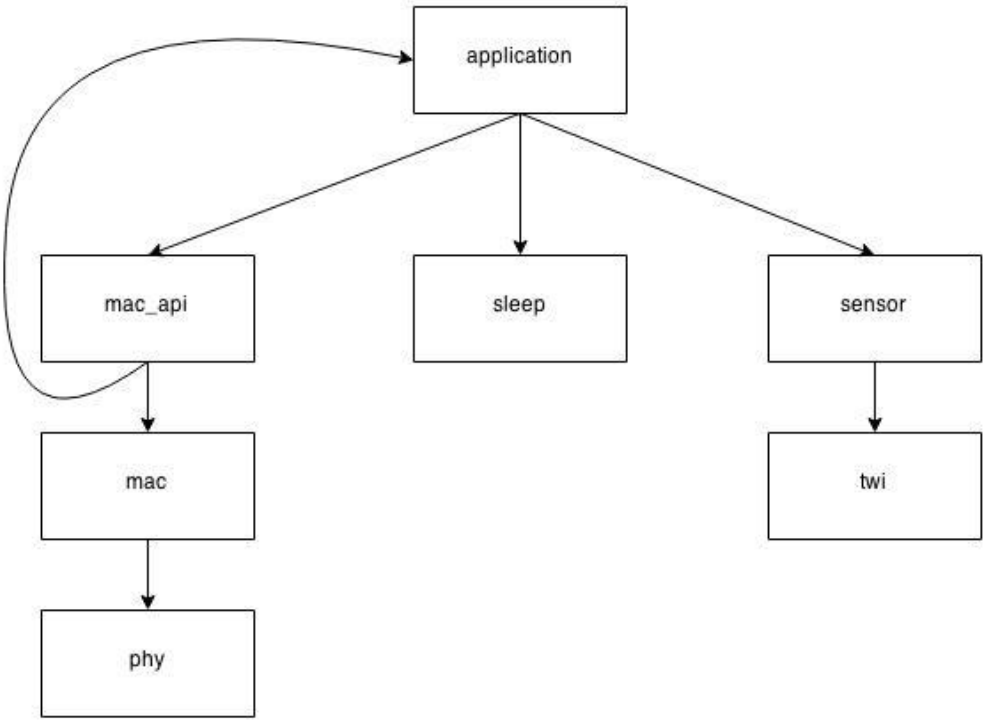

Figura 55. dependencias del programa de aplicación de los nodos finales.

8.3.1. COMUNICACIÓN

ZIGBEE

El programa de aplicación del dispositivo final implementa dos tipos de funciones para establecer comunicación con el coordinador. Entre ellas están las funciones que solicitan un servicio de la capa MAC y las funciones de callback que son llamadas desde la capa MAC cuyo cuerpo está definido en la aplicación.

El procedimiento de asociación, iniciado por un dispositivo final, consiste en la solicitud de asociación a través de la función de wpan_mlme_associate_req(). Una vez enviada la solicitud, se activa un temporizador para esperar a que el coordinador procese la solicitud de asociación. Una vez que expira el temporizado se envía una solicitud de transmisión de datos al coordinador para que responda a la solicitud de asociación previamente enviada. De nuevo, se activa un temporizador para esperar la respuesta del coordinador que es notificada a la capa de aplicación a través de la función usr_mlme_associate_conf(). La figura 56 muestra los eventos del procedimiento de asociación.

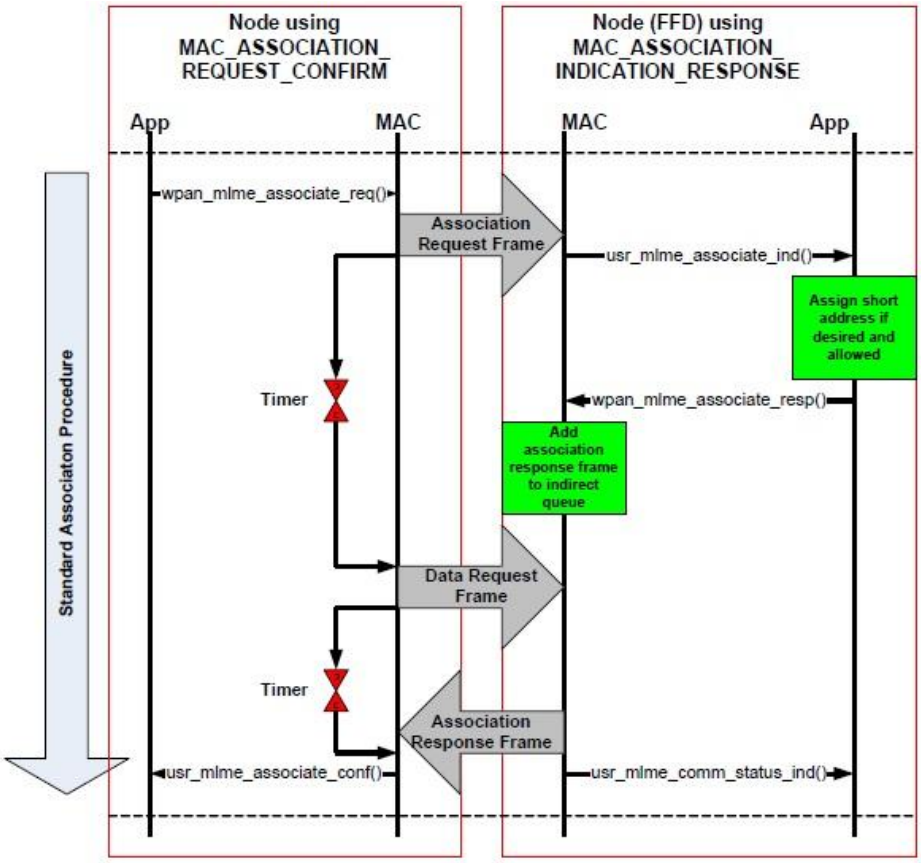

Figura 56. Eventos del procedimiento de asociación.

El procedimiento de desasociación puede ser iniciado tanto por el coordinador como por el dispositivo final. Si es iniciado por el coordinador, la solicitud se recibe a través de la función usr_mlme_disassociate_ind(). Si la solicitud es iniciada por un dispositivo final, se envía una solicitud de desasociación mediante la función wpan_mlme_disassociate_req().

La transmisión de datos se realiza de forma directa al coordinador con origen en los nodos finales. Para realizar una transmisión se envía la solicitud de envío de datos a la capa MAC a través de la función wpan_mpcs_data_req() indicando la longitud del mensaje, la carga útil y la dirección corta del dispositivo final.

El código referente a la comunicación ZigBee se ha implementado en los ficheros app.h y app.c que pueden encontrarse en el ANEXO 15.2.

TWI

El protocolo TWI es utilizado por los nodos finales para establecer comunicación con el sensor de temperatura digital AT30TSE758. El maestro en la comunicación es el

microcontrolador, mientras que el sensor es el esclavo con dirección 10010110. Los tres últimos bits de la dirección están definidos por la tensión de los pines A2, A1 y A0 del sensor. En este caso, A2 y A1 están a +VCC y A0 a GND por lo que sus valores digitales son 110 respectivamente.

La comunicación es controlada mediante la interrupción generada al finalizar un evento de lectura del buffer o escritura. Los registros relacionados con el módulo TWI, en concreto el registro de datos del buffer solo puede ser modificado dentro del ámbito de la rutina de interrupción. Si estos registros son modificados, al finalizar la interrupción se realiza la acción correspondiente y al finalizar la misma, vuelve a generarse una interrupción.

A través del registro de estado de la comunicación es conocido el punto exacto en el que se encuentra la misma, realizando así la acción adecuada que puede ser:

- Generar la señal de start para iniciar la comunicación.
- Enviar el byte de dirección formado por la dirección de esclavo del sensor en los 7 bits más significativos y el tipo de operación en el bit menos significativo (0 = escritura, 1 = lectura).
- Escribir en el registro “puntero a registros” del sensor, cuya dirección es 00h e indica el registro donde se van a realizar las posteriores operaciones de lectura o escritura.
- Escribir datos en el registro al que apunta el puntero.
- Leer del registro al que apunta el puntero.
- Generar la señal de stop para detener la comunicación.

Después de realizar cada operación de escritura de datos en un registro, se debe volver a escribir en el registro “puntero a registros” para continuar escribiendo más datos. En caso de lectura esto no es necesario.

El código referente a la comunicación TWI se ha implementado en los ficheros twi.h y twi.c que pueden encontrarse en el ANEXO 15.2.

8.3.2. AHORRO DE ENERGÍA

Una de las características principales de las redes ZigBee es la capacidad de los nodos de permanecer en un estado de bajo consumo o Sleep para optimizar el consumo de energía. En la tabla x. están los diferentes modos de sleep que puede adoptar el microcontrolador cuya fuente de interrupción para despertar difiere en cada uno de ellos.

En este caso, se va a implementar esta funcionalidad en los dispositivos finales, ya que el coordinador no puede permanecer dormido por el hecho de tener que alimentar el LCD. Teniendo en cuenta que el coordinador no transmitirá datos a los nodos finales, no se puede implementar un modo sleep en el que la fuente de interrupción sea la recepción de datos, sino que el nodo debe poder despertarse automáticamente tras un tiempo transcurrido parametrizado.

El modo de sleep que adoptaran los nodos finales es el "POWER-SAVE" en el que permanece activo el reloj periférico del timer/counter2 posibilitando que el nodo se despierte cuando se genere una interrupción por desbordamiento. El temporizador debe activarse antes de dormir al nodo, ya que después no se podrá y el nodo quedaría dormido indefinidamente.

El timer/counter2 es un temporizador de 8 bits con un reloj periférico de 1KHz con el preescalado al máximo. Esto significa que la interrupción por desbordamiento se produce cada $x = \frac{1}{1000} \cdot 255 = 255\text{ms}$, que es un intervalo de tiempo muy corto si se quiere utilizar el temporizador como fuente de interrupción para despertar el nodo. Debe permanecer dormido durante 30s entre las transmisiones de los datos de temperatura, por lo que se concatenan varios períodos de sleep de 255ms hasta alcanzar los 30s y el nodo despierte. El estado del nodo se indica con el led conectado al pin 4 del PUERTO E del microcontrolador permaneciendo apagado mientras el nodo está dormido y encendido mientras está despierto.

El código referente al control del modo sleep se ha implementado en el fichero app.c que puede encontrarse en el ANEXO 15.2.

9. RESULTADOS

El proyecto realizado cumple los objetivos marcados al inicio del mismo, habiendo conseguido diseñar en implementar una red en ZigBee de tres nodos con las siguientes características:

- La red es creada correctamente por el coordinador, permitiendo la asociación y desasociación de dispositivos.

- Las señales de control del LCD se generan correctamente consiguiendo un alto contraste entre los segmentos que deben estar iluminados y los que no.

- El método de control de la interfaz de usuario es sencillo e intuitivo con una velocidad de respuesta elevada para proporcionar una experiencia agradable para el usuario.

- La resolución de la temperatura mostrada en el LCD es de 1°C, utilizando unos márgenes de histéresis para que el valor mostrado no varíe demasiado deprisa cuando la temperatura oscila en el umbral de cambio.

- El consumo en los dispositivos finales es de 7.90mAh cuando el nodo se encuentra despierto y de 6.50mAh mientras está dormido. Por otro lado, el coordinador tiene un consumo de 10.20mAh. Estos valores han sido obtenidos midiendo con un amperímetro en serie en los pines que proporciona la plataforma ATmega256RFR2 para este fin.

- El rango de la red es de aproximadamente 8 metros en interiores y de 25 metros en línea de vista para un PER del 2%. De esta forma, si se sitúa el coordinador en el centro, podría cubrirse una zona de 201m^2 en interiores y de 1963.5m^2 en línea de vista. Para determinar el rango de la red se ha utilizado el indicador de calidad LQI que indica la calidad de un paquete recibido. Este indicador toma valores entre 0-255, donde 255 representa la máxima calidad. La relación entre el indicador LQI con la probabilidad de error del paquete se muestra en la figura 57.

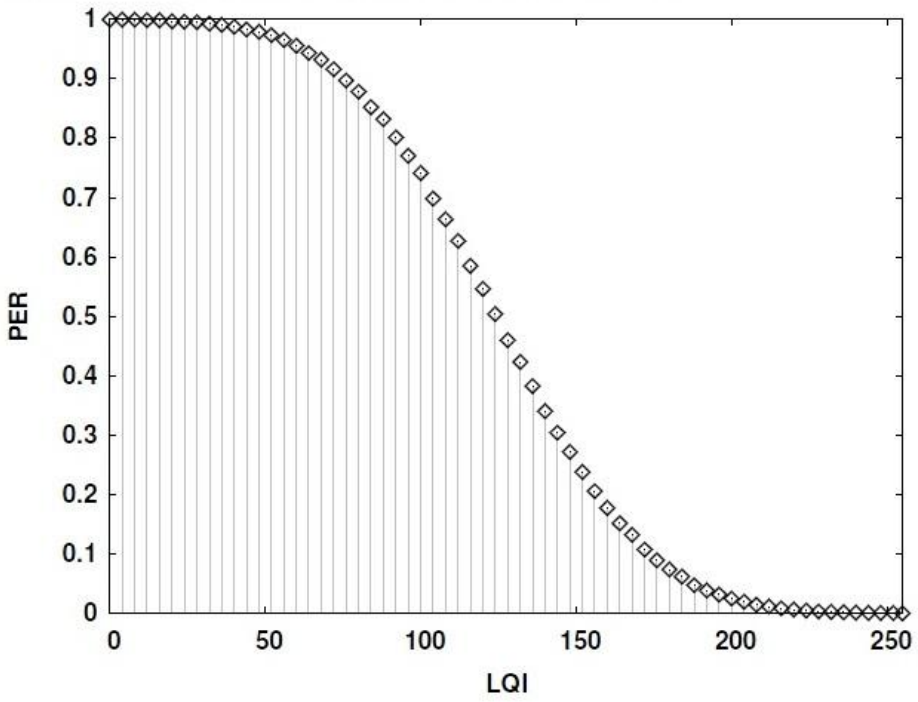

Figura 57. tasa de paquetes erróneos en función del indicador de calidad del enlace.

Como se puede ver en la figura, la tasa de error aumenta considerablemente a partir de un valor de LQI menor a 200, es decir, un 80% del valor de calidad máxima para el cual el 2% de los paquetes recibidos tendrá algún bit erróneo. Por lo tanto, se ha calculado la distancia para la cual el valor de calidad del enlace mostrado en el LCD es 80.

- La duración de las baterías, en los dispositivos finales es de aproximadamente 55 horas, pero, a medida que se reduce la capacidad la potencia transmitida por los dispositivos finales es menor. Para una distancia de 5m entre el coordinador y un dispositivo final, la potencia recibida por el coordinador es menor a la sensibilidad del receptor a las 21 horas. Para determinar la autónoma de los nodos se han realizado pruebas monitorizando el número de paquetes recibidos de un mismo nodo a través de un PC. Conocido este dato y sabiendo que los dispositivos finales transmiten una trama cada 30s se puede conocer el tiempo que ha durado la comunicación: $2572 \cdot \frac{30(\text{segundos}) \cdot 1(\text{hora})}{3600(segundos)} = 21.43\text{h}$. Realizando la misma prueba para el coordinador, se ha comprobado que tiene una autonomía de 14 horas.

La figura 58 muestra una captura de los tres nodos que forman la red ZigBee con los módulos de alimentación conectados a cada uno de ellos y el LCD conectado al coordinador.

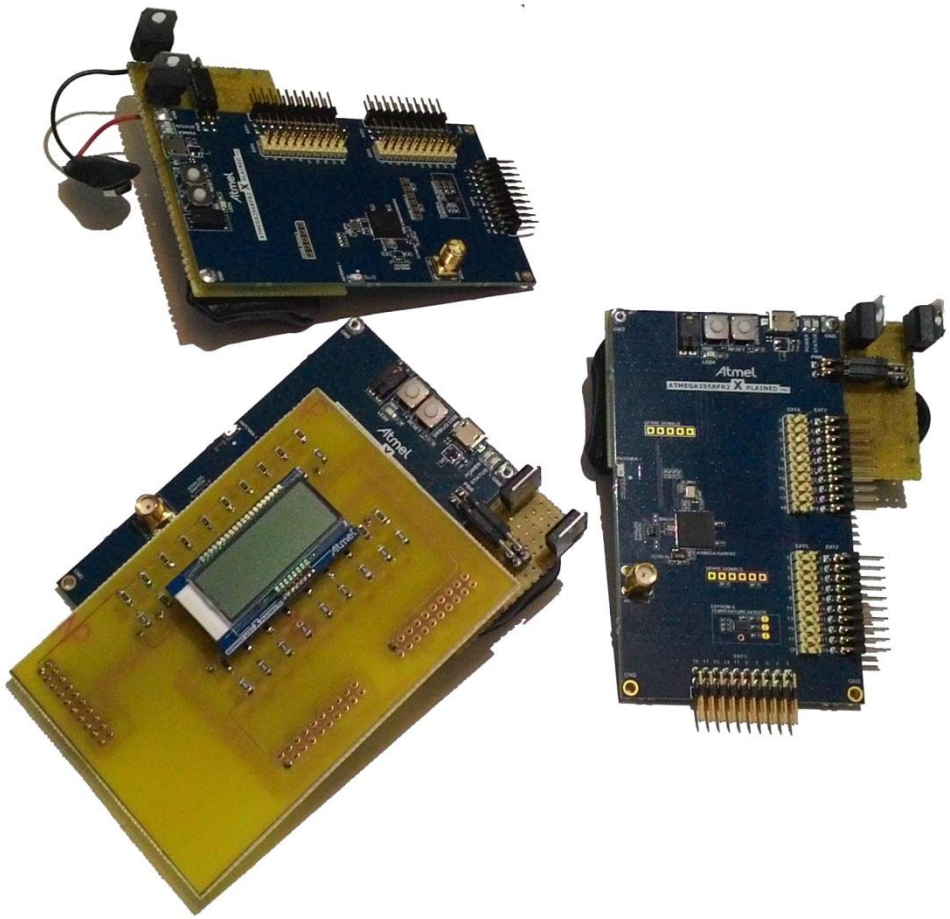

Figura 58. Captura de los nodos que forman la red ZigBee.

10. LINEAS FUTURAS DE DESARROLLO

A la vista de los resultados obtenidos, sería interesante desarrollar algunos aspectos para que pueda ser utilizado en un abanico de aplicaciones más amplio proporcionando soluciones de mayor calidad. En este apartado se va a hacer referencia a dichos aspectos indicando lo que aportaría.

- **Mejora de la eficiencia de los reguladores**: los reguladores utilizados en este proyecto son lineales, por lo que la relación de la potencia entregada al dispositivo y la disipada en el mismo no es muy elevada. El cambio a reguladores conmutados conllevaría una mejora en la autonomía de los dispositivos.

- **Incorporar una capa de red**: La incorporación de una capa de red al software de los dispositivos permitiría la integración de routers en la red, proporcionando la capacidad de extender de rango de la misma, robustez debido a la posibilidad de auto reparación encontrando rutas alternativas ante la caída de un enlace y la posibilidad de utilizar una topología mallada o en árbol.

- **Desarrollar la aplicación**: El programa de aplicación desarrollado en este proyecto, podría ser un punto de partida para una aplicación más ambiciosa en la que los datos recibidos por los nodos finales fuesen utilizados para comunicar emergencias, controlar actuadores o se almacenasen en una base de datos.

- **Incorporar una antena de mayor ganancia**: Para aumentar la distancia entre nodos sería conveniente incorporar una antena de mayor ganancia en el conector SMA de la plataforma. Su utilización provocaría un aumento del consumo, por lo que se debería conmutar el uso de ambas antenas de forma que los nodos utilizasen la antena de cerámica mientras el dispositivo se encuentre en estado IDLE y la antena de mayor ganancia para transmitir y recibir tramas de datos.

11. CONCLUSION

The project has been very useful to improve my skills as a software developer in the C programming language for microcontrollers, helping me to understand the architecture of the software used in communication standards through the study and understanding of the software stack used in this project and expand my knowledge of embedded software development. Furthermore, in this project I learned to make designs of printed circuit boards and put into practice electronic knowledge learned during the career.

As to the objectives set at the beginning of the project, all of them have been met with good results, although parts as consumption of the devices could be improved with hardware and software changes.

The chosen technology has helped to develop the project and meet the objectives set at the beginning satisfactorily. Has managed to form a three-node network with a star topology using the ZigBee protocol that allows temperature monitoring at the position where the end devices are placed and the characteristic parameters of the network. The printed circuit board allows to connect the LCD screen to the microcontroller so it can be controlled by software, controlling the data shown on the screen using the button on the platform. Power modules perform the required functions, provides the energy needed for the platforms and ensures the necessary security to protect electronic components. In terms of software, has been designed a robust and computationally efficient architecture that is able to process all tasks required for proper system operation.

12. GLOSARIO

A efectos de este trabajo, se aplican los siguientes términos y definiciones.

Asociación: Procedimiento por el cual un dispositivo final se incorpora a la red.

Baliza: Trama de datos utilizada por el coordinador de red para sincronizarse con el resto de dispositivos. Las balizas pueden ser transmitidas de forma periódica o ante solicitud de un dispositivo final.

Brecha digital: desigualdad entre las personas que pueden tener acceso o conocimiento en relación a las nuevas tecnologías y las que no.

Coordinador: Dispositivo principal de la red encargado de su creación, gestión y mantenimiento. Es el maestro en la sincronización con el resto de dispositivos en redes balizadas y gestiona la asociación y desasociación de nodos.

Depuración: Proceso en el que el programa es analizado con ayuda de una serie de herramientas que permiten monitorizarlo con el objetivo de corregir fallos.

Diagrama de flujo: Representación gráfica de un proceso que relaciona mediante flechas los distintos pasos del mismo por medio de símbolos que contienen una breve descripción del estado en el que se encuentra el proceso.

Dispositivo final: Dispositivo de funcionalidad reducida que se encuentra asociado a la red y que puede comunicarse con el coordinador si la topología de la red es en estrella o con cualquier dispositivo si se trata de una red mallada o en árbol.

Driver: Software que se encuentra en el nivel más bajo del programa y que controla los registros de los módulos de hardware para proporcionar un servicio básico que permita a las capas superiores utilizar dicho hardware.

Entidad: Bloque de software que se encarga de gestionar un aspecto determinado del programa y proporciona una serie de servicios relacionados dentro de una arquitectura organizada.

Footprint: Es la representación gráfica que el programa de diseño de placas de circuito impreso hace de la vista en “planta” de un encapsulado. Proporciona la información necesaria para ensamblar el componente a la placa.

Función de callback: Función utilizada por un fichero o normalmente por una biblioteca mediante un puntero a función proporcionado como argumento en la inicialización de dicho fichero o biblioteca cuyo cuerpo está definido en un fichero que pertenece a una capa superior

dentro de la arquitectura del programa. Este tipo de funciones son muy útiles en bibliotecas para evitar dependencias de la aplicación en las funciones definidas dentro de la misma, permitiendo definirlas de forma externa.

Gateway: Pasarela de datos entre dos protocolos de comunicación distintos. En este proyecto se habla de pasarela entre ZigBee

Getter: Función que permite conocer el valor de variables privadas de un fichero de forma global a través de los parámetros de la misma.

Histéresis: En el contexto de este proyecto, la histéresis representa la tendencia de la temperatura representada en el LCD a conservar su nivel actual para evitar que el valor representado oscile rápidamente entre dos valores. De esta forma se consigue un intervalo de cambio del valor de la temperatura en vez de un umbral fijo, como ocurre con algunas propiedades de ciertos materiales.

Internet de las cosas: es el concepto de comunicación por el cual todos los objetos que nos rodean se encuentran conectados permanentemente a Internet, permitiendo la captura, almacenamiento y gestión de toda la información emitida por dichos objetos con la finalidad de automatizar actividades y procesos diarios en nuestra vida cotidiana así como analizar todos los datos generados para mejorar la toma de decisiones en situaciones que se presentan cada día.

Interrupción: Evento generado por el módulo de interrupciones de un microcontrolador mediante el cual se detiene la ejecución del programa de aplicación para ejecutar la función de interrupción correspondiente. Una vez que la rutina de interrupción es ejecutada, el hilo del programa vuelve al punto en el que se encontraba antes de producirse la interrupción.

Mapeo: Procedimiento por el cual se asignan valores a una serie de variables en función de un criterio determinado.

Mecanismo CSMA-CA: Es un mecanismo de acceso al medio compartido mediante escucha de portadora en el canal. Si el canal se encuentra ocupado, vuelve a intentar la transmisión una vez transcurrido un tiempo aleatorio.

Modo sleep: Modo de bajo consumo del microcontrolador en el que se encuentran la mayor parte del tiempo los dispositivos finales de una red ZigBee para aumentar la duración de las baterías.

Nodo: Cada uno de los dispositivos que forman la red.

Perfil: Particularización del protocolo ZigBee para una aplicación concreta.

Poll: Procedimiento por el cual un dispositivo solicita la transmisión de datos pendientes a otro dispositivo. Es utilizado para la transmisión de tramas de forma indirecta.

Puntero a registro: Registro que contiene la dirección de memoria otro registro donde se van a realizar las operaciones de lectura o escritura solicitadas por el maestro en una comunicación.

Resistencia de pull-up: Resistencia situada en los pines de I/O del microcontrolador que puede ser habilitada por software cuando el pin se encuentra configurado como entrada.

Setter: Función que permite modificar variables privadas de un fichero de forma global a través de los parámetros de la misma.

Stack: Es el software de un estándar o protocolo de comunicación. Implementa las capas del protocolo y las interfaces entre las mismas, proporcionando unos servicios espeficificados por el estándar o protocolo a la capa de aplicación del programa.

Transceptor: dispositivo que realiza funciones tanto de envío como de recepción de señales, empleando elementos comunes del circuito para ambas funciones

.

13. ACRÓNIMOS Y ABREVIATURAS

ASF	Atmel Software Framework
ASK	Amplitude Shift Keying
BPSK	Binary Phase-Shift Keying
CAP	Contention Access Period
CCA	Clear Channel Assessment
CFP	Contention-Free Period
CPU	Central Processing Unit
CSMA-CA	Carrier Sense Multiple Access – Collision Avoidance
ED	Energy Detection
EDBG	Embedeed Debuger
EEPROM	Electronically Erasable Programmable Read Only Memory
FFD	Full Function Device
FSTN	Film compensated Super Twisted Nematic
GND	Ground
GPIO	General Purpose Input Output
GTS	Guaranteed Time Slot
ISM	Industrial, Specific and Metrical band
ISP	Integrated System Peripheral
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LOS	Line Of Shight

LQI	Link Quality Indicator
MAC	Medium Access Control
MCL	MAC Core Layer
MCPS	MAC common part sublayer
MCU	Microcontroller Unit
MIPS	Million Instructions Per Second
MLME	MAC Layer Management Entity
MPDU	MAC protocol data unit
NLDE	Network Layer Data Entity
O-QPSK	Offset Quadrature Phase-Shift Keying
PAL	Platform Abstraction Layer
PAN	Personal Area Network
PCB	Printed Circuit Board
PD	Physical Data
PHY	Physical layer
PLME	Physical Layer Management Entity
PPDU	PHY protocol data unit
PWM	Pulse Width Modulation
PX	Port X
RAM	Random-Access Memory
ROM	Read-Only Memory
RFD	Reduced Function Device
SAP	Service Acces Point
SMA	SubMiniature version A

SFD	Spreading Factor Detection
SPI	Serial Peripheral Interface
TAL	Transceiver Abstraction Layer
TWI	Two Wires Interface
UART	Universal Asynchronous Receiver-Transmitter
USART	Universal Synchronous Asynchronous Receiver-Transmitter
VCC	Voltage at the Common Collector

14. BIBLIOGRAFÍA

[1] The Institute of Electrical and Electronics Engineers, **Inc*. IEEE Std 802.15.4k™-2013*, [En linea]**. **New York (USA), 14 de agosto de 2013. Disponible en: <http://standards.ieee.org/getieee802/download/ 802.15.4k2013.pdf.>** [Consulta: 15-08-2014].

[2] ZigBee Alliance, Inc. *ZigBee Specification,* [En linea]. Disponible en: <http://home.deib.polimi.it/cesana/teaching/IoT/papers/ZigBee/ ZigBeeSpec.pdf>. [Consulta: 15-08-2014].

[3] Ata Elahi, Adam Gschwender. *ZigBee Wireless Sensor and Control Network,* [En linea]. Pearson Education, 29 de oct. de 2009. ISBN : 013705940X, 9780137059409. [Consulta: 20-08-2014].

[4] Atmel Corporation. *ATmega256/128/64RFR2 Wireless MCU*, [En linea]. Disponible en: <http://www.atmel.com/Images/Atmel-8393- MCU_Wireless-ATmega256RFR2-ATmega128RFR2 ATmega64RFR2_Datasheet.pdf>. [Consulta: 28-08-2014].

[5] Atmel Corporation. *Atmel ATMEGA256RFR2 Xplained Pro [USER GUIDE],* [En linea]. Disponible en: <http://www.atmel.com/Images/Atmel-42079-ATMEGA256RFR2-Xplained-Pro_User-Guide.pdf>. [Consulta: 02-09-2014].

[6] Atmel Corporation. *Atmel ATMEGA256RFR2 Xplained Pro Design Documentation,* [En linea]. Disponible en: http://www.atmel.com/Images/ Atmel-42079-ATMEGA256RFR2-XplainedPro_User-Guide.zip.> [Consulta: 02-09-2014].

[7] Atmel Corporation. *Atmel Segment LCD1 Xplained Pro [USER GUIDE],* [En linea]. Disponible en: <http://www.atmel.com/images/atmel-42076-segment-lcd1-xplained-pro_user-guide.pdf>. [Consulta: 03-09-2014].

[8] Atmel Corporation.. ***Atmel AVR2025: IEEE 802.15.4 MAC Software Package - User Guide,* [En linea]. Disponible en: <http://www.atmel.com/images/ doc8412.pdf>.** [Consulta: 15-10-2014].

15. ANEXOS

15.1. CÓDIGO FUENTE DEL COORDINADOR

```
/*-----------------------------------------------------------------------------
 - FILE: app.h                                                                -
 -----------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                              -
 -----------------------------------------------------------------------------
 - DESCRIPTION: Header file which contains the definitions, enumerations,
   structures and function prototypes used in app.c file.
 -----------------------------------------------------------------------------*/


#ifndef APP_H_
#define APP_H_

#include "ui.h"


/*****************************************************/
/*  DEFINITIONS                                      */
/*****************************************************/

#define MAX_NUMBER_OF_DEVICES           (2)


/*****************************************************/
/*  STRUCTURES & ENUMERATIONS                        */
/*****************************************************/

/* stored information of a node */
typedef struct app_node_info_t{
   uint8_t  nodeID;
   uint16_t shortAddress;

   int16_t  temperature;
   uint8_t  tempDec;
   uint8_t  tempUnit;

   /* quality in percent */
   uint8_t  linkQuality;
   uint8_t  lqDec;
   uint8_t  lqUnit;

   uint16_t txPackets;
   uint8_t  tpDec;
   uint8_t  tpUnit;

   }APP_NODE_INFO_T;

/* context of the application */
typedef struct app_ctx_t{
   APP_NODE_INFO_T nodeInfo[MAX_NUMBER_OF_DEVICES];
   UI_STATE_E uiState;
   uint16_t   uiUpdateCounter;
   }APP_CTX_T;
```

```
#endif /* APP_H_ */



/*-------------------------------------------------------------------------
 - FILE: app.c                                                            -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
 - DESCRIPTION: This file contains the main function and implements the
   application program.
 -------------------------------------------------------------------------*/


#include <string.h>
#include <inttypes.h>
#include <stdio.h>
#include <avr/io.h>
#include "avr2025_mac.h"
#include "temp.h"
#include "SLCD_Driver.h"
#include <asf.h>
#include "app.h"
#include "ui.h"
#include "button.h"

/************************************************/
/*  DEFINITIONS                                 */
/************************************************/

#define CHANNEL_OFFSET                  (2)
#define LED_PIN                         LED_ON_BOARD



/************************************************************************/
/* GLOBAL VARIABLES                                                     */
/************************************************************************/

associated_device_t device_list[MAX_NUMBER_OF_DEVICES];
uint16_t no_of_assoc_devices;
APP_CTX_T GLB_appCtx;
coord_state_t coord_state = COORD_STARTING;
static uint32_t tx_cnt;
uint8_t current_channel;
uint8_t current_channel_page;
static uint32_t channels_supported;



/************************************************************************/
/* STATIC FUNCTION PROTOTYPES                                           */
/************************************************************************/
static bool assign_new_short_addr(uint64_t addr64, uint16_t *addr16);
static void app_init(void);
static void app_task(void);
static void storeNodeData(uint8_t *msdu,
             uint8_t msduLength,
             uint8_t mpduLinkQuality,
             uint16_t short_address);

static void app_ui(UI_STATE_E uiState);



/************************************************************************
```

```
* FUNCTION:
*
* main
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* This function initializes application, the MAC, initiates a MLME reset
* request, and implements the main loop of the stack and the application.
*****************************************************************************/
int main(void)
{
   app_init  ();
   irq_initialize_vectors();
   sw_timer_init();
   cpu_irq_enable();
   LED_Off(LED_PIN);
   DDRB |= 16;

   wpan_mlme_reset_req(true);

   while (true) {

      wpan_task();
      app_task();
   }
}

/*****************************************************************************
* FUNCTION:
*
* usr_mcps_data_ind
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @SrcAddrSpec:      Pointer to source address specification
* @DstAddrSpec:      Pointer to destination address specification
* @msduLength:       Number of octets contained in MSDU
* @msdu:             Pointer to MSDU
* @mpduLinkQuality:  LQI measured during reception of the MPDU
* @DSN:              DSN of the received data frame.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* Callback function usr_mcps_data_ind. This function is executed by the
* MAC layer when a frame is received.
*****************************************************************************/

void usr_mcps_data_ind(wpan_addr_spec_t *SrcAddrSpec,
     wpan_addr_spec_t *DstAddrSpec,
     uint8_t msduLength,
     uint8_t *msdu,
     uint8_t mpduLinkQuality,
     uint8_t DSN,){

   storeNodeData(msdu, msduLength,
             mpduLinkQuality,
             SrcAddrSpec->Addr.short_address);

}

/*****************************************************************************
* FUNCTION:
*
* usr_mlme_associate_ind
```

```
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @DeviceAddress: Extended address of device requesting association
* @CapabilityInformation: Capabilities of device requesting association
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* Callback function usr_mlme_associate_ind. This function is executed by
* the MAC layer when the coordinator receive an association request from
* an end device.
*****************************************************************************/
void usr_mlme_associate_ind(uint64_t DeviceAddress,
     uint8_t CapabilityInformation)
{


   uint16_t associate_short_addr = macShortAddress_def;

   if (assign_new_short_addr(DeviceAddress,
        &associate_short_addr) == true) {

      wpan_mlme_associate_resp(DeviceAddress,
                               associate_short_addr,
                               ASSOCIATION_SUCCESSFUL);

      LED_On(LED_PIN);
      GLB_appCtx.nodeInfo[no_of_assoc_devices-1].shortAddress =
      associate_short_addr;

      GLB_appCtx.nodeInfo[no_of_assoc_devices-1].nodeID = no_of_assoc_devices;

   } else {
      wpan_mlme_associate_resp(DeviceAddress,
                               associate_short_addr,
                               PAN_AT_CAPACITY);
   }
}


/*****************************************************************************
* FUNCTION:
*
* assign_new_short_addr
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @addr64: long address of the end device which want associate.
* @addr16: short address of the end device which want associate.
*
* OUT:
* @bool: returns true if a valid short address has been found, and false
* if not.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* Application specific function to assign a short address for a new end
* device associated.
*****************************************************************************/
static bool assign_new_short_addr(uint64_t addr64, uint16_t *addr16)
{
   uint8_t i;
```

```
   /* Check if device has been associated before */
   for (i = 0; i < MAX_NUMBER_OF_DEVICES; i++) {
      if (device_list[i].short_addr == 0x0000) {
         /* If the short address is 0x0000, it has not been used
          * before */
         continue;
      }

      if (device_list[i].ieee_addr == addr64) {
         /* Assign the previously assigned short address again */
         *addr16 = device_list[i].short_addr;
         return true;
      }
   }

   for (i = 0; i < MAX_NUMBER_OF_DEVICES; i++) {
      if (device_list[i].short_addr == 0x0000) {
         *addr16 = CPU_ENDIAN_TO_LE16(i + 0x0001);
         /* get next short address */
         device_list[i].short_addr = CPU_ENDIAN_TO_LE16(i + 0x0001);
         /* store extended address */
         device_list[i].ieee_addr = addr64;
         no_of_assoc_devices++;

         return true;
      }
   }

   /* the network is at full capacity, not assign address */
   return false;
}



/*************************************************************************
* FUNCTION:
*
* app_init
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* initializes the application context variables to 0 and calls all the
* initialization functions of the program.
*************************************************************************/
static void app_init(void){

   uint8_t i;

   for(i = 0; i < MAX_NUMBER_OF_DEVICES; i++){
      GLB_appCtx.nodeInfo[i].shortAddress = 0;
      GLB_appCtx.nodeInfo[i].nodeID = 0;
      GLB_appCtx.nodeInfo[i].linkQuality = 0;
      GLB_appCtx.nodeInfo[i].lqDec = 0;
      GLB_appCtx.nodeInfo[i].lqUnit = 0;
      GLB_appCtx.nodeInfo[i].temperature = 0;
      GLB_appCtx.nodeInfo[i].tempDec = 0;
      GLB_appCtx.nodeInfo[i].tempUnit = 0;
      GLB_appCtx.nodeInfo[i].txPackets = 0;
      GLB_appCtx.nodeInfo[i].tpDec = 0;
      GLB_appCtx.nodeInfo[i].tpUnit = 0;
   }

   GLB_appCtx.uiState = UI_STATE_MENU_TR;
   GLB_appCtx.uiUpdateCounter = 0;

   SLCD_init();
```

```
   TEMP_init();
   buttonInit();
   uiInit();
}



/*************************************************************************
* FUNCTION:
*
* app_task
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* handles the updating of the LCD
*************************************************************************/
static void app_task(void){

   app_ui(GLB_appCtx.uiState);
}



/*************************************************************************
* FUNCTION:
*
* storeNodeData
*************************************************************************/
/*************************************************************************
* PARAMETERS:
*
* IN:
* @SrcAddrSpec:      source address of the received data.
* @msduLength:       Number of octets contained in MSDU
* @msdu:             Pointer to MSDU
* @mpduLinkQuality:  LQI measured during reception of the MPDU
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* this function store the current temperature value of each node in the
* network.
*************************************************************************/
static void storeNodeData(uint8_t *msdu,
                          uint8_t msduLength,
                          uint8_t mpduLinkQuality,
                          uint16_t short_address){

   uint8_t i;

   for(i = 0; i < no_of_assoc_devices; i++){
      if(GLB_appCtx.nodeInfo[i].shortAddress == short_address){
         break;
      }
   }

   GLB_appCtx.nodeInfo[i].temperature = TEMP_update (msdu, i);

   GLB_appCtx.nodeInfo[i].tempDec     =
   (GLB_appCtx.nodeInfo[i].temperature / 10);

   GLB_appCtx.nodeInfo[i].tempUnit    =
   (GLB_appCtx.nodeInfo[i].temperature % 10) + 10;

   GLB_appCtx.nodeInfo[i].linkQuality =
```

```
   (((mpduLinkQuality + 170) * 100) / 255);

   GLB_appCtx.nodeInfo[i].lqDec       =
   (GLB_appCtx.nodeInfo[i].linkQuality / 10);

   GLB_appCtx.nodeInfo[i].lqUnit      =
   (GLB_appCtx.nodeInfo[i].linkQuality % 10);

   if(GLB_appCtx.nodeInfo[i].linkQuality >= 100){
      GLB_appCtx.nodeInfo[i].lqDec  = 9;
      GLB_appCtx.nodeInfo[i].lqUnit = 9;
   }

   GLB_appCtx.nodeInfo[i].txPackets++;
   GLB_appCtx.nodeInfo[i].tpDec       =
   (GLB_appCtx.nodeInfo[i].txPackets / 10);

   GLB_appCtx.nodeInfo[i].tpUnit      =
   (GLB_appCtx.nodeInfo[i].txPackets % 10);
   }


/*****************************************************************************
* FUNCTION:
*
* app_ui
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @uiState: state of the user interface state machine.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* calls SLCD_update to represent the data in the LCD depending of the
* current state of the user interface state machine.
*****************************************************************************/
static void app_ui(UI_STATE_E uiState){

   switch(uiState){
      case UI_STATE_MENU_TR:
         SLCD_update (20, 20, 26, 22);
      break;

      case UI_STATE_MENU_TE:
         SLCD_update (20, 20, 26, 21);
      break;

      case UI_STATE_MENU_CE:
         SLCD_update (20, 20, 25, 21);
      break;

      case UI_STATE_MENU_PT:
         SLCD_update (20, 20, 24, 26);
      break;

      case UI_STATE_TR:
         SLCD_update (20,
                      20,
                      (((uint8_t)no_of_assoc_devices+1)/10),
                      (((uint8_t)no_of_assoc_devices)+1) % 10);
      break;

      case UI_STATE_TE_NODE1:
         SLCD_update (20,
                       1,
```

```
                        GLB_appCtx.nodeInfo[0].tempDec,
                        GLB_appCtx.nodeInfo[0].tempUnit);
      break;

      case UI_STATE_TE_NODE2:
         SLCD_update (20,
                        2,
                        GLB_appCtx.nodeInfo[1].tempDec,
                        GLB_appCtx.nodeInfo[1].tempUnit);
      break;

      case UI_STATE_CE_NODE1:
         SLCD_update (20,
                        1,
                        GLB_appCtx.nodeInfo[0].lqDec,
                        GLB_appCtx.nodeInfo[0].lqUnit);
      break;

      case UI_STATE_CE_NODE2:
         SLCD_update (20,
                        2,
                        GLB_appCtx.nodeInfo[1].lqDec,
                        GLB_appCtx.nodeInfo[1].lqUnit);
      break;

      case UI_STATE_PT_NODE1:
         SLCD_update (20,
                        1,
                        GLB_appCtx.nodeInfo[0].tpDec,
                        GLB_appCtx.nodeInfo[0].tpUnit);
      break;

      case UI_STATE_PT_NODE2:
         SLCD_update (20,
                        2,
                        GLB_appCtx.nodeInfo[1].tpDec,
                        GLB_appCtx.nodeInfo[1].tpUnit);
      break;

      default:
      break;
   }
}



/*****************************************************************************
* FUNCTION:
*
* ISR of TIMER3_OVF_vect.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* handles the button event detection and the updating of the user
* interface state machine every 20ms.
*****************************************************************************/
 ISR(TIMER3_OVF_vect){

   GLB_appCtx.uiState = uiUpdate();
}
```

```
/*-------------------------------------------------------------------------
 - FILE: temp.h                                                           -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
- DESCRIPTION: Header file which contains the definitions, enumerations,
   structures and function prototypes used in temp.c file.
 -------------------------------------------------------------------------*/


#ifndef TEMP_H_
#define TEMP_H_

#include "asf.h"
#include "app.h"
#include <avr/io.h>
#include <stdio.h>

/************************************************/
/*  DEFINITIONS                                 */
/************************************************/

#define TEMP_THRESHOLD       8
#define TEMP_HYSTERESIS      5


/************************************************/
/*  STRUCTURES & ENUMERATIONS                   */
/************************************************/

typedef enum bool_e{

   FALSE = 0,
   TRUE
   }BOOL_E;

typedef struct node_temp_t{
    int16_t   readData;
    int16_t   currentTempData;
    int16_t   currentTemp;
    BOOL_E    firstRead;
   }NODE_TEMP_T;


typedef struct sensor_ctx_t{
   NODE_TEMP_T nodeTemp[MAX_NUMBER_OF_DEVICES];
   }SENSOR_CTX_T;


/************************************************/
/*  FUNCTION PROTOTYPES                         */
/************************************************/

void TEMP_init     (void);
int16_t TEMP_update     (uint8_t *pData, uint8_t nodeNumber);


#endif /* TEMP_H_ */
```

```
/*-------------------------------------------------------------------------
 - FILE: temp.c                                                           -
```

```
 ------------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                              -
 ------------------------------------------------------------------------------
 - DESCRIPTION: This file contains a couple of functions to manage and store
   the temperature data received of each end device in the network.
 ------------------------------------------------------------------------------*/

#include "temp.h"
#include "SLCD_driver.h"


/*****************************************************************************/
/* GLOBAL VARIABLES                                                          */
/*****************************************************************************/

SENSOR_CTX_T  sensor_ctx;
const uint8_t threshold  = TEMP_THRESHOLD;
const uint8_t hysteresis = TEMP_HYSTERESIS;


/*****************************************************************************
* FUNCTION:
*
* TEMP_init
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* this function initializes temperature variables to 0 and first read
* boolean to TRUE.
*****************************************************************************/
void TEMP_init(void){
   uint8_t i;

   for(i = 0; i < MAX_NUMBER_OF_DEVICES; i++){
      sensor_ctx.nodeTemp[i].currentTemp = 0;
      sensor_ctx.nodeTemp[i].currentTempData = 0;
      sensor_ctx.nodeTemp[i].readData = 0;
      sensor_ctx.nodeTemp[i].firstRead = TRUE;
   }

}



/*****************************************************************************
* FUNCTION:
*
* TEMP_update
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* OUT:
* @sensor_ctx.nodeTemp[nodeNumber].currentTemp: current temperature of
* a node to be stored.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* this function converts the temperature data received to celsius and
* increments, decrements or does not change the temperature stored for
* this node depending of its value.
*****************************************************************************/
int16_t TEMP_update  (uint8_t *pData, uint8_t nodeNumber){

   uint8_t sign = 1;
   uint8_t data_received[2];
```

```
   data_received[0] = *(pData);
   data_received[1] = *(pData + 1);

   sensor_ctx.nodeTemp[nodeNumber].readData = (int16_t)data_received[0];

   sensor_ctx.nodeTemp[nodeNumber].readData =
   sensor_ctx.nodeTemp[nodeNumber].readData << 8;

   sensor_ctx.nodeTemp[nodeNumber].readData =
   (sensor_ctx.nodeTemp[nodeNumber].readData) |
   (int16_t)(data_received[1]);


   /* absolute value. */
   if (sensor_ctx.nodeTemp[nodeNumber].readData & (1 << 15)){
      sign *= -1;
      sensor_ctx.nodeTemp[nodeNumber].readData &= ~(1 << 15);
   }

   /* converts data to celsius  for a resolution of 12 bits. */
   sensor_ctx.nodeTemp[nodeNumber].readData =
   (sensor_ctx.nodeTemp[nodeNumber].readData >> 4);

   if(sensor_ctx.nodeTemp[nodeNumber].firstRead){

      sensor_ctx.nodeTemp[nodeNumber].currentTemp =
      sensor_ctx.nodeTemp[nodeNumber].readData * sign * 0.0625;

      sensor_ctx.nodeTemp[nodeNumber].currentTempData =
      sensor_ctx.nodeTemp[nodeNumber].currentTemp / 0.0625;

      sensor_ctx.nodeTemp[nodeNumber].firstRead = FALSE;

      return sensor_ctx.nodeTemp[nodeNumber].currentTemp;

   }else{
      /* received date temp is higher than current data temp +
       *Threshold + hysteresis */
      if((sensor_ctx.nodeTemp[nodeNumber].readData) >
      (sensor_ctx.nodeTemp[nodeNumber].currentTempData +
       threshold + hysteresis)){

        sensor_ctx.nodeTemp[nodeNumber].currentTemp++;
        sensor_ctx.nodeTemp[nodeNumber].currentTempData =
        sensor_ctx.nodeTemp[nodeNumber].currentTemp / 0.0625;

        return sensor_ctx.nodeTemp[nodeNumber].currentTemp;

      /* received date temp is lower than current data temp -
       *Threshold - hysteresis */
      }else if((sensor_ctx.nodeTemp[nodeNumber].readData) <
       (sensor_ctx.nodeTemp[nodeNumber].currentTempData -
       threshold - hysteresis)){

        sensor_ctx.nodeTemp[nodeNumber].currentTemp--;
        sensor_ctx.nodeTemp[nodeNumber].currentTempData =
        sensor_ctx.nodeTemp[nodeNumber].currentTemp / 0.0625;

        return sensor_ctx.nodeTemp[nodeNumber].currentTemp;
      }
   }

   return sensor_ctx.nodeTemp[nodeNumber].currentTemp;
}
```

```
/*-------------------------------------------------------------------------
 - FILE: ui.h                                                             -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
- DESCRIPTION: Header file which contains the enumerations, structures and
   function prototypes used in ui.c file.
 -------------------------------------------------------------------------*/

#ifndef UI_H_
#define UI_H_



/*************************************************/
/*  STRUCTURES & ENUMERATIONS                    */
/*************************************************/

typedef enum ui_state_e{

   UI_STATE_MENU_TR = 0,
   UI_STATE_MENU_CE,
   UI_STATE_MENU_PT,
   UI_STATE_MENU_TE,
   UI_STATE_TR,
   UI_STATE_CE_NODE1,
   UI_STATE_CE_NODE2,
   UI_STATE_PT_NODE1,
   UI_STATE_PT_NODE2,
   UI_STATE_TE_NODE1,
   UI_STATE_TE_NODE2
   }UI_STATE_E;

typedef struct ui_ctx_t{

   UI_STATE_E uiState;
   }UI_CTX_T;



/*************************************************/
/*  FUNCTION PROTOTYPES                          */
/*************************************************/

void        uiInit    (void);
UI_STATE_E  uiUpdate  (void);


#endif /* UI_H_ */
```

```
/*-------------------------------------------------------------------------
 - FILE: ui.c                                                             -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
 - DESCRIPTION: This file implements the behavior of the user interface state
   machine. Poll button events every 20ms for update its state.
 -------------------------------------------------------------------------*/

#include "ui.h"
#include "button.h"
```

```
/*****************************************************************************/
/* GLOBAL VARIABLES                                                          */
/*****************************************************************************/

UI_CTX_T uiCtx;


/*****************************************************************************
* FUNCTION:
*
* uiInit
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* initializes the registers of timer/counter3 which will generate an
* interruption every 20ms to poll the button and update the user
* interface state machine.
*****************************************************************************/
void uiInit (void){

   TIMSK3 = 0x01;
   TCCR3B = 0x01;

   uiCtx.uiState = UI_STATE_MENU_TR;
}


/*****************************************************************************
* FUNCTION:
*
* uiUpdate
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* OUT:
* @uiCtx.uiState: current state of the user interface state machine.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* this function updates the user interface state machine calling
* buttonUpdate for get button events every 20ms.
*****************************************************************************/
UI_STATE_E uiUpdate (void){

   BUTTON_EVENT_E event = buttonUpdate();

   switch(uiCtx.uiState){

      case UI_STATE_MENU_TR:
      if(event == BUTTON_EVENT_PRESS_SHORT){
         uiCtx.uiState = UI_STATE_MENU_CE;
      }else if(event == BUTTON_EVENT_PRESS_LONG){
         uiCtx.uiState = UI_STATE_TR;
      }
      break;

      case UI_STATE_MENU_CE:
      if(event == BUTTON_EVENT_PRESS_SHORT){
         uiCtx.uiState = UI_STATE_MENU_PT;
         }else if(event == BUTTON_EVENT_PRESS_LONG){
         uiCtx.uiState = UI_STATE_CE_NODE1;
      }
      break;
```

```
case UI_STATE_MENU_PT:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_MENU_TE;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_PT_NODE1;
}
break;

case UI_STATE_MENU_TE:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_MENU_TR;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_TE_NODE1;
}
break;


case UI_STATE_TR:
if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_TR;
}
break;

case UI_STATE_CE_NODE1:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_CE_NODE2;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_CE;
}
break;

case UI_STATE_CE_NODE2:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_CE_NODE1;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_CE;
}
break;

case UI_STATE_PT_NODE1:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_PT_NODE2;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_PT;
}
break;

case UI_STATE_PT_NODE2:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_PT_NODE1;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_PT;
}
break;

case UI_STATE_TE_NODE1:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_TE_NODE2;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_TE;
}
break;

case UI_STATE_TE_NODE2:
if(event == BUTTON_EVENT_PRESS_SHORT){
   uiCtx.uiState = UI_STATE_TE_NODE1;
   }else if(event == BUTTON_EVENT_PRESS_LONG){
   uiCtx.uiState = UI_STATE_MENU_TE;
```

```
      }
      break;


      default:
      break;
      }

      return uiCtx.uiState;
}



/*-------------------------------------------------------------------------
 - FILE: button.h                                                        -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                         -
 -------------------------------------------------------------------------
- DESCRIPTION: Header file which contains the enumerations, structures and
   function prototypes used in button.c file.
 -------------------------------------------------------------------------*/
#ifndef BUTTON_H_
#define BUTTON_H_


#include <stdio.h>
#include <avr/io.h>



/*************************************************/
/*  STRUCTURES & ENUMERATIONS                    */
/*************************************************/

#define REG_BIT_4_SHIFT  4;


typedef enum button_event_e{

   BUTTON_EVENT_NONE = 0,
   BUTTON_EVENT_PRESS_SHORT,
   BUTTON_EVENT_PRESS_LONG
   }BUTTON_EVENT_E;


typedef enum button_state_e{

   BUTTON_STATE_UNKNOWN = 0,
   BUTTON_SATATE_UNPRESSED,
   BUTTON_STATE_REBOUNDS,
   BUTTON_STATE_PRESS_SHORT,
   BUTTON_STATE_PRESS_LONG
   }BUTTON_STATE_E;


typedef struct button_ctx_t{

   uint8_t counter;
   BUTTON_STATE_E butState;
   }BUTTON_CTX_T;


/*************************************************/
/*  FUNCTION PROTOTYPES                          */
/*************************************************/

void        buttonInit      (void);
```

```
BUTTON_EVENT_E  buttonUpdate    (void);

#endif /* BUTTON_H_ */



/*-----------------------------------------------------------------------------
 - FILE: button.c                                                             -
 -----------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                              -
 -----------------------------------------------------------------------------
 - DESCRIPTION: This file implements a functions for detect push  button
   events by polling the input register in which the button is connected every
   20ms.
 -----------------------------------------------------------------------------*/

#include "button.h"

/*****************************************************************************/
/* GLOBAL VARIABLES                                                          */
/*****************************************************************************/

BUTTON_CTX_T swButton;


/*****************************************************************************/
/* STATIC FUNCTION PROTOTYPES                                                */
/*****************************************************************************/

static uint8_t getPinState(void);



/*****************************************************************************
* FUNCTION:
*
* buttonInit
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* initializes the pin 4 of port E as input because is where the button
* is connected.
*****************************************************************************/
void buttonInit (void){
   DDRE  &= 0xEF;

   swButton.butState = BUTTON_STATE_UNKNOWN;
   swButton.counter  = 0;
}



/*****************************************************************************
* FUNCTION:
*
* buttonUpdate
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* OUT:
* @event: event detected in the button.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
```

```
*
* this function detects short and long presses in the button doing poll
* to the button state once every 20ms.
*****************************************************************/
BUTTON_EVENT_E buttonUpdate (void){

   BUTTON_EVENT_E event = BUTTON_EVENT_NONE;

   switch(swButton.butState){

      case BUTTON_STATE_UNKNOWN:
      case BUTTON_SATATE_UNPRESSED:

         swButton.counter = 0;

         if(getPinState() == 0){
            swButton.butState = BUTTON_STATE_REBOUNDS;
            swButton.counter++;
            }else{
            swButton.butState = BUTTON_SATATE_UNPRESSED;
         }
      break;

      case BUTTON_STATE_REBOUNDS:
      if(swButton.counter < 5){
         swButton.counter++;
      }else{
         if(getPinState() == 0){
            swButton.butState = BUTTON_STATE_PRESS_SHORT;
            }else{
            swButton.butState = BUTTON_SATATE_UNPRESSED;
         }
      }
      break;

      case BUTTON_STATE_PRESS_SHORT:

         if(getPinState() == 0){
            swButton.counter++;
         }else{
            swButton.butState = BUTTON_SATATE_UNPRESSED;
            event = BUTTON_EVENT_PRESS_SHORT;
         }

         if(swButton.counter > 26){
            swButton.butState = BUTTON_STATE_PRESS_LONG;
            event = BUTTON_EVENT_PRESS_LONG;
         }
      break;

      case BUTTON_STATE_PRESS_LONG:
         if(getPinState() == 1){
            swButton.butState = BUTTON_SATATE_UNPRESSED;
         }
      break;
   }

   return event;
}



/*****************************************************************************
* FUNCTION:
*
* getPinState
*****************************************************************************/
/*****************************************************************************
```

```
* PARAMETERS:
*
* OUT:
* @value: current register value of pin 4 of port E. returns 0 if button
* is pressed and 1 if button isn't pressed.
*************************************************************************/
/************************************************************************
* DESCRIPTION:
*
* return the button state pressed or unpressed.
*************************************************************************/
static uint8_t getPinState(void){
   uint8_t value;
   value = (PINE & 0x10) >> REG_BIT_4_SHIFT;

   return value;
}
```

```
/*-------------------------------------------------------------------------
 - FILE: SLCD_Driver.h                                                    -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
- DESCRIPTION: Header file which contains the definitions, enumerations,
   structures and  function prototypes used in SLCD_Driver.c file.
 -------------------------------------------------------------------------*/


#ifndef SLCD_DRIVER_H_
#define SLCD_DRIVER_H_

#include <avr/io.h>
#include <stdio.h>



/*************************************************/
/*  DEFINITIONS                                  */
/*************************************************/

#define CHARACTER_NUM_PINS     4
#define NUM_BITS               40
#define COM_0                  0x08
#define COM_1                  0x04
#define COM_2                  0x02
#define COM_3                  0x01
#define LSB_MASK               0x01


typedef enum SLCD_cicle_e{

   CYCLE_1 = 0,
   CYCLE_2,
   CYCLE_3,
   CYCLE_4,
   REST_CYCLE,

   CYCLO_MAX
   }SLCD_CYCLE;
```

```
/*************************************************/
/*  STRUCTURES & ENUMERATIONS                    */
/*************************************************/

typedef struct caracteres {

   uint8_t character_1;
   uint8_t character_2;
   uint8_t character_3;
   uint8_t character_4;
}  IN_CHARACTERS;


typedef struct patterns {

   uint8_t pattern_1[4];
   uint8_t pattern_2[4];
   uint8_t pattern_3[4];
   uint8_t pattern_4[4];
}  CHARACTER_PATTERNS;


typedef struct cicle{

   uint8_t port_B[8];
   uint8_t PB_dec;
   uint8_t port_D[8];
   uint8_t PD_dec;
   uint8_t port_E[8];
   uint8_t PE_dec;
   uint8_t port_F[8];
   uint8_t PF_dec;
   uint8_t port_G[8];
   uint8_t PG_dec;
}  DUTY_CYCLE;


typedef struct bits{

   DUTY_CYCLE first_cycle;
   DUTY_CYCLE second_cycle;
   DUTY_CYCLE third_cycle;
   DUTY_CYCLE fourth_cycle;
}  PORT_BITS;


/*************************************************/
/*  FUNCTION PROTOTYPES                          */
/*************************************************/

extern void    SLCD_init   (void);
extern void    SLCD_deInit (void);
extern void    SLCD_update
                   (uint8_t CARACTER_1,
                    uint8_t CARACTER_2,
                   uint8_t CARACTER_3,
                   uint8_t CARACTER_4);


#endif /* SLCD_DRIVER_H_ */
```

```
/*-------------------------------------------------------------------------
 - FILE: SLCD_Driver.c                                                    -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
 - DESCRIPTION: This file implements the functions nedded to control the I/O
   registers of the pins connected to the LCD providing the service to control
   the LCD with an only function.
 -------------------------------------------------------------------------*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include "SLCD_Driver.h"

/*************************************************************************/
/* GLOBAL VARIABLES                                                      */
/*************************************************************************/

IN_CHARACTERS        characters;
CHARACTER_PATTERNS   paterns;
PORT_BITS            bits;
uint8_t              phase;

/*************************************************************************/
/* STATIC FUNCTION PROTOTYPES                                            */
/*************************************************************************/

static void           bitMap            (void);
static void           assignPatterns    (void);
static void           binToDec          (void);
static uint8_t        conversion        (uint8_t port_bits[8]);
static uint8_t        powerOfTwo        (uint8_t exponent);


/*************************************************************************/
/* CONSTANTS                                                             */
/*************************************************************************/


static const uint8_t pattern[122] =
             /*< in number in SLCD_update()>  < represented character> */
{0x00,0x07,0x0e,0x00,  /*<  0  >                        <    0  > */
 0x00,0x00,0x06,0x00,  /*<  1  >                        <    1  > */
 0x02,0x03,0x0c,0x04,  /*<  2  >                        <    2  > */
 0x02,0x01,0x0e,0x04,  /*<  3  >                        <    3  > */
 0x02,0x04,0x06,0x04,  /*<  4  >                        <    4  > */
 0x02,0x05,0x0a,0x04,  /*<  5  >                        <    5  > */
 0x02,0x07,0x0a,0x04,  /*<  6  >                        <    6  > */
 0x00,0x00,0x0e,0x00,  /*<  7  >                        <    7  > */
 0x02,0x07,0x0e,0x04,  /*<  8  >                        <    8  > */
 0x02,0x05,0x0e,0x04,  /*<  9  >                        <    9  > */
 0x00,0x07,0x0f,0x00,  /*< 10  >                        <  0°C  > */
 0x00,0x00,0x07,0x00,  /*< 11  >                        <  1°C  > */
 0x02,0x03,0x0d,0x04,  /*< 12  >                        <  2°C  > */
 0x02,0x01,0x0f,0x04,  /*< 13  >                        <  3°C  > */
 0x02,0x04,0x07,0x04,  /*< 14  >                        <  4°C  > */
 0x02,0x05,0x0b,0x04,  /*< 15  >                        <  5°C  > */
 0x02,0x07,0x0b,0x04,  /*< 16  >                        <  6°C  > */
```

```
 0x00,0x00,0x0f,0x00,  /*< 17  >                    <  7°C  > */
 0x02,0x07,0x0f,0x04,  /*< 18  >                    <  8°C  > */
 0x02,0x05,0x0f,0x04,  /*< 19  >                    <  9°C  > */
 0x00,0x00,0x00,0x00,  /*< 20  >                    <  esp  > */
 0x02,0x07,0x08,0x04,  /*< 21  >                    <    E  > */
 0x03,0x06,0x0c,0x04,  /*< 22  >                    <    R  > */
 0x02,0x06,0x0e,0x04,  /*< 23  >                    <    A  > */
 0x02,0x06,0x0c,0x04,  /*< 24  >                    <    P  > */
 0x00,0x07,0x08,0x00,  /*< 25  >                    <    C  > */
 0x08,0x00,0x08,0x01}; /*< 26  >                    <    T  > */
```

```
/*****************************************************************************
* FUNCTION:
*
* ISR of TIMER0_OVF_vect
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* interrupt routine which handles I/O register values to generate SEG
* and COM signals.
*****************************************************************************/
ISR (TIMER0_OVF_vect){

   uint8_t cycle = phase / 2;
   uint8_t PB    = 0;
   uint8_t PD    = 0;
   uint8_t PE    = 0;
   uint8_t PF    = 0;
   uint8_t PG    = 0;

   switch(cicle){

      case CYCLE_1:
      PB = bits.first_cycle.PB_dec;
      PD = bits.first_cycle.PD_dec;
      PE = bits.first_cycle.PE_dec;
      PF = bits.first_cycle.PF_dec;
      PG = bits.first_cycle.PG_dec;
      break;

      case CYCLE_2:
      PB = bits.second_cycle.PB_dec;
      PD = bits.second_cycle.PD_dec;
      PE = bits.second_cycle.PE_dec;
      PF = bits.second_cycle.PF_dec;
      PG = bits.second_cycle.PG_dec;
      break;

      case CYCLE_3:
      PB = bits.third_cycle.PB_dec;
      PD = bits.third_cycle.PD_dec;
      PE = bits.third_cycle.PE_dec;
      PF = bits.third_cycle.PF_dec;
      PG = bits.third_cycle.PG_dec;
      break;

      case CYCLE_4:
      PB = bits.fourth_cycle.PB_dec;
      PD = bits.fourth_cycle.PD_dec;
      PE = bits.fourth_cycle.PE_dec;
      PF = bits.fourth_cycle.PF_dec;
      PG = bits.fourth_cycle.PG_dec;
      break;
```

```
      }

      if(cycle < REST_CYCLE){
         if(phase % 2 == 0){

            PORTB = (PORTB & 0x11) | (PB);
            DDRB  = (DDRB  & 0x11) | (PB);
            PORTD = PD ^ 64;
            DDRD  = PD;
            PORTE = (PORTE & 0x1C) | (PE ^ 192);
            DDRE  = (DDRE  & 0x1C) | (PE);
            PORTF = PF;
            DDRF  = PF;
            PORTG = PG ^ 4;
            DDRG  = PG;

         }else{

            PORTB = (PORTB & 0x11) | (PB  ^ 238);
            DDRB  = (DDRB  & 0x11) | (PB);
            PORTD = PD ^ 176;
            DDRD  = PD;
            PORTE = (PORTE & 0x1C) | (PE ^ 35);
            DDRE  = (DDRE  & 0x1C) | (PE);
            PORTF = PF ^ 8;
            DDRF  = PF;
            PORTG = PG ^ 33;
            DDRG  = PG;
         }

      }else{

         DDRB  = (DDRB  & 0x11) | 0xEE;
         DDRD  = 0xFF;
         DDRE  = (DDRE  & 0x1C) | 0xE3;
         DDRF  = 0xFF;
         DDRG  = 0xFF;
         PORTB = (PORTB & 0x11);
         PORTD = 0;
         PORTE = (PORTE & 0x1C);
         PORTF = 0;
         PORTG = 0;

      }

   phase++;

   if(phase > 9){

      phase = 0;
   }
}


/*****************************************************************************
* FUNCION:
*
* SLCD_init
*****************************************************************************/
/*****************************************************************************
* DESCRIPCION:
*
* Initializes the I/O registers and enables global interrupts.
*****************************************************************************/
extern void SLCD_init(void){

   sei();
```

```
   DDRB  = (DDRB  & 0x11) | 0xEE;
   DDRD  = (DDRD  & 0x0F) | 0xF0;
   DDRE  = (DDRE  & 0x1C) | 0xE3;
   DDRF  = (DDRF  & 0xF7) | 0x08;
   DDRG  = (DDRG  & 0xAD) | 0x52;
   PORTB = (PORTB & 0x11);
   PORTD = (PORTD & 0x0F);
   PORTE = (PORTE & 0x1C);
   PORTF = (PORTF & 0xF7);
   PORTG = (PORTG & 0xAD);

   TCCR0B = 0x02;
   TIMSK0 = 0x01;
   TCNT0  = 0x00;

   phase = 0;
}



/*****************************************************************************
* FUNCION:
*
* SLCD_deInit
*****************************************************************************/
/*****************************************************************************
* DESCRIPCION:
*
* clear all registers and variables used in SLCD_Driver
*****************************************************************************/
extern void SLCD_deInit(void){

   DDRB  = (DDRB  & 0x11) | 0xEE;
   DDRD  = (DDRD  & 0x0F) | 0xF0;
   DDRE  = (DDRE  & 0x1C) | 0xE3;
   DDRF  = (DDRF  & 0xF7) | 0x08;
   DDRG  = (DDRG  & 0xAD) | 0x52;
   PORTB = (PORTB & 0x11);
   PORTD = (PORTD & 0x0F);
   PORTE = (PORTE & 0x1C);
   PORTF = (PORTF & 0xF7);
   PORTG = (PORTG & 0xAD);

   TIMSK0 = 0x00;
   TCNT0  = 0x00;

   phase = 0;
}


/*****************************************************************************
* FUNCION:
*
* powerOfTwo
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN
* @exponent: exponent of two.
*
* OUT:
* @result: result of the power of two of the exponent passed as argument.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* Calculates the power of two with the exponent passed as argument.
```

```
*
*****************************************************************************/
static uint8_t powerOfTwo (uint8_t exponent){
   uint8_t i;
   uint8_t result = 1;

   for(i=0; i<exponent; i++){
      result *= 2;
   }
   return result;
}


/*****************************************************************************
* FUNCTION:
*
* conversion.
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @bits_port[8]: array which contains the 8 bits of an port
*
* OUT:
* @result: Decimal value of the byte
*****************************************************************************/
/*****************************************************************************
* DESCRIPCION:
*
* Converts the 8 bits of a register to decimal value.
*
*****************************************************************************/
static uint8_t conversion(uint8_t bits_port[8]){
   uint8_t i;
   uint8_t result = 0;
   for(i=0; i<8; i++){
      if(bits_port[i] == 1){
         result += powerOfTwo(i);
      }
   }
   return result;
}


/*****************************************************************************
* FUNCTION:
*
* binToDec
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* calls conversion function to convert every 8 register bits to decimal.
*****************************************************************************/
static void binToDec (void){

   bits.first_cycle.PB_dec  = conversion (bits.first_cycle.port_B);
   bits.first_cycle.PD_dec  = conversion (bits.first_cycle.port_D);
   bits.first_cycle.PE_dec  = conversion (bits.first_cycle.port_E);
   bits.first_cycle.PF_dec  = conversion (bits.first_cycle.port_F);
   bits.first_cycle.PG_dec  = conversion (bits.first_cycle.port_G);

   bits.second_cycle.PB_dec = conversion (bits.second_cycle.port_B);
   bits.second_cycle.PD_dec = conversion (bits.second_cycle.port_D);
   bits.second_cycle.PE_dec = conversion (bits.second_cycle.port_E);
   bits.second_cycle.PF_dec = conversion (bits.second_cycle.port_F);
   bits.second_cycle.PG_dec = conversion (bits.second_cycle.port_G);
```

```
   bits.third_cycle.PB_dec  = conversion (bits.third_cycle.port_B);
   bits.third_cycle.PD_dec  = conversion (bits.third_cycle.port_D);
   bits.third_cycle.PE_dec  = conversion (bits.third_cycle.port_E);
   bits.third_cycle.PF_dec  = conversion (bits.third_cycle.port_F);
   bits.third_cycle.PG_dec  = conversion (bits.third_cycle.port_G);

   bits.fourth_cycle.PB_dec  = conversion (bits.fourth_cycle.port_B);
   bits.fourth_cycle.PD_dec  = conversion (bits.fourth_cycle.port_D);
   bits.fourth_cycle.PE_dec  = conversion (bits.fourth_cycle.port_E);
   bits.fourth_cycle.PF_dec  = conversion (bits.fourth_cycle.port_F);
   bits.fourth_cycle.PG_dec  = conversion (bits.fourth_cycle.port_G);
}


/*****************************************************************************
* FUNCTION:
*
* assignPatterns
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* assign the character pattern for each characters.characterX integer
* value.
*****************************************************************************/
static void assignPatterns(void){

   uint8_t i;

   for (i=0 ; i<CHARACTER_NUM_PINS; i++){

      patterns.pattern_1[i] = pattern[characters.character_1 * 4 + i];
      patterns.pattern_2[i] = pattern[characters.character_2 * 4 + i];
      patterns.pattern_3[i] = pattern[characters.character_3 * 4 + i];
      patterns.pattern_4[i] = pattern[characters.character_4 * 4 + i];
   }
}


/*****************************************************************************
* FUNCTION:
*
* bitMap
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* map the bits of each I/O port with the character patterns.
*****************************************************************************/
static void bitMap (void){

   uint8_t i;
   assignPatterns();

   uint8_t bitMap[40] = {[ 1] = patterns.pattern_2 [3],
                         [ 2] = patterns.pattern_4 [2],
                         [ 3] = patterns.pattern_2 [1],
                         [ 5] = patterns.pattern_3 [1],
                         [ 6] = patterns.pattern_3 [0],
                         [ 7] = patterns.pattern_1 [0],
                         [12] = patterns.pattern_1 [3],
                         [13] = patterns.pattern_3 [2],
                         [14] = COM_2,
                         [15] = patterns.pattern_2 [2],
                         [16] = patterns.pattern_4 [1],
                         [17] = patterns.pattern_4 [0],
                         [21] = patterns.pattern_3 [3],
```

```
                        [22] = COM_0,
                        [23] = COM_1,
                        [27] = patterns.pattern_1 [2],
                        [32] = patterns.pattern_4 [3],
                        [34] = COM_3,
                        [37] = patterns.pattern_2 [0]};

   for(i=0; i<NUM_BITS; i++){
      if(i<8){          /* Port B bits*/

         bits.first_cycle.port_B [i]    =bitMap [i] >> 3;
         bits.second_cycle.port_B[i]    =(bitMap[i] >> 2) & LSB_MASK;
         bits.third_cycle.port_B [i]    =(bitMap[i] >> 1) & LSB_MASK;
         bits.fourth_cycle.port_B [i]   =bitMap [i]       & LSB_MASK;

      }else if(i<16){  /* Port D bits */

         bits.first_cycle.port_D [i-8]  =bitMap [i] >> 3;
         bits.second_cycle.port_D[i-8]  =(bitMap[i] >> 2) & LSB_MASK;
         bits.third_cycle.port_D [i-8]  =(bitMap[i] >> 1) & LSB_MASK;
         bits.fourth_cycle.port_D [i-8] =bitMap [i]       & LSB_MASK;

      }else if(i<24){  /* Port E bits */

         bits.first_cycle.port_E [i-16] =bitMap [i] >> 3;
         bits.second_cycle.port_E[i-16] =(bitMap[i] >> 2) & LSB_MASK;
         bits.third_cycle.port_E [i-16] =(bitMap[i] >> 1) & LSB_MASK;
         bits.fourth_cycle.port_E [i-16]=bitMap [i]       & LSB_MASK;

      }else if(i<32){  /* Port F bits */

         bits.first_cycle.port_F [i-24] =bitMap [i] >> 3;
         bits.second_cycle.port_F[i-24] =(bitMap[i] >> 2) & LSB_MASK;
         bits.third_cycle.port_F [i-24] =(bitMap[i] >> 1) & LSB_MASK;
         bits.fourth_cycle.port_F [i-24]=bitMap [i]       & LSB_MASK;

      }else if(i<40){  /* Port G bits */

         bits.first_cycle.port_G[i-32] =bitMap [i] >> 3;
         bits.second_cycle.port_G[i-32]=(bitMap[i] >> 2) & LSB_MASK;
         bits.third_cycle.port_G[i-32] =(bitMap[i] >> 1) & LSB_MASK;
         bits.fourth_cycle.port_G[i-32]=bitMap [i]       & LSB_MASK;
      }
   }
}


/*****************************************************************************
* FUNCTION:
*
* SLCD_update
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @character1: character which we want to represent in the first LCD
* character
* @character2: character which we want to represent in the second LCD
* character
* @character3: character which we want to represent in the fourth LCD
* character
* @character4: character which we want to represent in the fifth LCD
* character
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
```

```
* this function is used by the application for represent the characters
* passed as arguments.
*****************************************************************************/
extern void SLCD_update (uint8_t character1,
            uint8_t character2,
            uint8_t character3,
            uint8_t character4)
{
   characters.character_1 = character1;
   characters.character_2 = character2;
   characters.character_3 = character3;
   characters.character_4 = character4;

   bitMap   ();
   binToDec ();
}
```

15.2. CÓDIGO FUENTE DE LOS DISPOSITIVOS FINALES

```
/*-----------------------------------------------------------------------------
 - FILE: app.h                                                                -
 -----------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                              -
 -----------------------------------------------------------------------------
- DESCRIPTION: Header file which contains the enumerations and structures used
   in app.c file.
 -----------------------------------------------------------------------------*/

#include "avr2025_mac.h"


/*************************************************/
/*  STRUCTURES & ENUMERATIONS                    */
/*************************************************/

typedef struct app_transmission_info_t{

   uint8_t src_addr_mode;
   wpan_addr_spec_t dst_addr;
   uint8_t payload_len;
   uint8_t payload[2];
   uint8_t msduHandle;
}  APP_TRANSMISSION_INFO_T;
```

```
/*-----------------------------------------------------------------------------
 - FILE: app.c                                                                -
 -----------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                              -
 -----------------------------------------------------------------------------
- DESCRIPTION: This file contains the main function and implements the
   application program.
 -----------------------------------------------------------------------------*/


#include <string.h>
#include <inttypes.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include "sleepmgr.h"
#include "avr2025_mac.h"
#include "common_sw_timer.h"
#include "sensor.h"
#include "twi.h"
#include "pal.h"
#include "app.h"
#include <asf.h>


/*************************************************************************/
/*  DEFINITIONS                                                          */
/*************************************************************************/
```

```
/** Channel Offset will give us the channel number as (CHANNEL_OFFSET + 11) */
#define CHANNEL_OFFSET                  (2)
/** Defines the short scan duration time. */
#define SCAN_DURATION_SHORT             (5)
/** Defines the long scan duration time. */
#define SCAN_DURATION_LONG              (6)
/** Define the LED on duration time. */

#define TIMER_SYNC_BEFORE_ASSOC_MS      (3000)
#define APP_GUARD_TIME_US               (10000)
#define PAYLOAD_LEN                     (2)

#define LED_PIN                         LED_ON_BOARD


/*============================================================================*/
/* GLOBAL VARIABLES                                                           */
/*============================================================================*/

uint16_t readyForSleepCount = 0;
/* This structure stores the short and extended address of the coordinator. */
wpan_addr_spec_t coord_addr_spec;
/** This variable stores the current state of the node. */
app_state_t app_state = APP_IDLE;
static uint8_t APP_TIMER;
uint8_t sleepCounter = 0;
bool nodeSleep   = false;
bool transmissionRequested = false;
APP_TRANSMISSION_INFO_T tInfo;

/** This array stores the current msdu payload. */
static uint8_t msdu_payload[PAYLOAD_LEN];

uint8_t current_channel;
uint8_t current_channel_page;
static uint32_t channels_supported;



/*============================================================================*/
/* STATIC FUNCTION PROTOTYPES                                                 */
/*============================================================================*/

static void network_search_indication_cb(void *parameter);
static void app_task(void);



/*****************************************************************************
* FUNCTION:
*
* main
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* This function initializes application, the MAC, initiates a MLME reset
* request, and implements the main loop of the stack and the application.
*****************************************************************************/
int main(void)
{

   app_init()

   wpan_mlme_reset_req(true);
```

```
   while (true) {

      wpan_task();

      if(nodeAssociated == 1){
         app_task();
      }
   }
}



/*************************************************************************
* FUNCTION:
*
* app_init
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* Initializes all the modules in the program and HW registers.
*************************************************************************/
static void app_init(void){

   twi_init();
   sensor_conf();

   irq_initialize_vectors();
   Enable_global_interrupt();

   /* sleep mode set as power save */
   volatile enum sleepmgr_mode mode = SLEEPMGR_PSAVE;

   /* initializes sleepmgr. */
   sleepmgr_init();
   sleepmgr_lock_mode(mode);

   /* timer 2 HW initialization */
   TCCR2B  = 0x07;
   TIMSK2  = 0x01;
   TCNT2   = 0x00;

   /* LED output initialization */
   DDRB |= 16;
   LED_On(LED_PIN);

   sw_timer_init();
   sw_timer_get_id(&APP_TIMER);
}



/*************************************************************************
* FUNCTION:
*
* usr_mlme_scan_conf
*************************************************************************/
/*************************************************************************
* PARAMETERS:
*
* IN:
* @param status            Result of requested scan operation
* @param ScanType          Type of scan performed
* @param ChannelPage       Channel page on which the scan was performed
* @param UnscannedChannels Bitmap of unscanned channels
* @param ResultListSize    Number of elements in ResultList
* @param ResultList        Pointer to array of scan results
*************************************************************************/
```

```
/************************************************************************
* DESCRIPTION:
*
* Callback function usr_mlme_scan_conf. This function is executed by the
* MAC layer after of scan all the channels.
************************************************************************/
void usr_mlme_scan_conf(uint8_t status,
      uint8_t ScanType,
      uint8_t ChannelPage,
      uint32_t UnscannedChannels,
      uint8_t ResultListSize,
      void *ResultList)
{
   if (status == MAC_SUCCESS) {
      wpan_pandescriptor_t *coordinator;
      uint8_t i;


      coordinator = (wpan_pandescriptor_t *)ResultList;

      for (i = 0; i < ResultListSize; i++) {
         /*
          * Check if the PAN descriptor belongs to our
          * coordinator.
          * Check if coordinator allows association.
          */
         if ((coordinator->LogicalChannel == current_channel) &&
               (coordinator->ChannelPage ==
               current_channel_page) &&
               (coordinator->CoordAddrSpec.PANId ==
               DEFAULT_PAN_ID) &&
               ((coordinator->SuperframeSpec &
               ((uint16_t)1 <<
               ASSOC_PERMIT_BIT_POS)) ==
               ((uint16_t)1 << ASSOC_PERMIT_BIT_POS))
               ) {
            /* Store the coordinator's address information.
            **/
            coord_addr_spec.AddrMode = WPAN_ADDRMODE_SHORT;
            coord_addr_spec.PANId = DEFAULT_PAN_ID;
            ADDR_COPY_DST_SRC_16(
                  coord_addr_spec.Addr.short_address,
                  coordinator->CoordAddrSpec.Addr.short_address);
            /* Set proper state of application. */
            app_state = APP_SCAN_DONE;

            uint16_t pan_id;
            pan_id = DEFAULT_PAN_ID;
            wpan_mlme_set_req(macPANId,
            return;
         }

         /* Get the next PAN descriptor. */
         coordinator++;
      }

      /*
       * If here, the result list does not contain our expected
       * coordinator.
       * Let's scan again.
       */
      wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
            SCAN_CHANNEL(current_channel),
            SCAN_DURATION_SHORT,
            current_channel_page);
   } else if (status == MAC_NO_BEACON) {
      /*
       * No beacon is received; no coordiantor is located.
```

```
       * Scan again, but used longer scan duration.
       */
      wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
           SCAN_CHANNEL(current_channel),
           SCAN_DURATION_LONG,
           current_channel_page);
   } else {
      /* Set proper state of application. */
      app_state = APP_IDLE;

      /* Something went wrong; restart. */
      wpan_mlme_reset_req(true);
   }
}


/*************************************************************************
* FUNCTION:
*
* app_task
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* handles the application behavior. Sends the temperature data to the
* coordinator once every 30 seconds, when the end device is awake. Then,
* sleeps the node again.
*************************************************************************/
static void app_task (void){

   uint8_t *pData;

   if (nodeSleep == true){
      return;
   /* if node is awake */
   }else{
        /* if transmission has not been requested yet*/
      if(transmissionRequested == false){

         /* configures the transmission*/
         tInfo.src_addr_mode = WPAN_ADDRMODE_SHORT;
         tInfo.dst_addr.AddrMode = WPAN_ADDRMODE_SHORT;
         tInfo.dst_addr.PANId = DEFAULT_PAN_ID;
         ADDR_COPY_DST_SRC_16(tInfo.dst_addr.Addr.short_address, coord_addr);
         tInfo.payload_len = 2;

         /* read the temperature from the sensor */
         pData = read_temperature ();
         tInfo.payload[0] = *(pData);
         tInfo.payload[1] = *(pData + 1);
         tInfo.msduHandle++;

         /* data transmission request*/
         wpan_mcps_data_req(tInfo.src_addr_mode,
         &tInfo.dst_addr,
         tInfo.payload_len,
         tInfo.payload,
         tInfo.msduHandle,
         WPAN_TXOPT_ACK);

         transmissionRequested = true;
      /* if transmission has been requested */
      }else{

         /* if the node is ready to sleep  */
         if (readyForSleepCount > 4000){

            readyForSleepCount = 0;
```

```
            transmissionRequested = false;
            nodeSleep = true;

            LED_Off(LED_PIN);
            sleepmgr_enter_sleep();
            LED_On(LED_PIN);
         }else{

            readyForSleepCount++;
         }
      }
   }
}


/*************************************************************************
* FUNCTION:
*
* ISR of TIMER2_OVF_vect.
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* timer/counter2 overflow interruption routine which handle power save
* status of the node. The interruption is generated once every 0.25
* seconds.
*************************************************************************/
ISR(TIMER2_OVF_vect)
{
   if(nodeSleep == true){
      /* wake up the node every 30 seconds*/
      if(sleepCounter > 120){

         nodeSleep    = false;
         sleepCounter = 0;
         }else{
         /* sleep the node again */
         sleepCounter++;
         sleepmgr_enter_sleep();
      }
   }
}
```

```
/*-------------------------------------------------------------------------
 - FILE: sensor.h                                                        -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                         -
 -------------------------------------------------------------------------
- DESCRIPTION: Header file which contains the definitions, enumerations,
   structures and  function prototypes used in sensor.c file.
 -------------------------------------------------------------------------*/


#ifndef SENSOR_H_
#define SENSOR_H_

#include "asf.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
#include <stdio.h>
#include <twi.h>


/*************************************************/
/*  DEFINITIONS                                  */
/*************************************************/

#define TWI_BASE_REG                  &TWBR
#define SLAVE_BUS_ADDR                0x96
#define SLAVE_MEM_ADDR_LENGTH         TWI_SLAVE_ONE_BYTE_SIZE
#define TWI_SPEED_HZ                  125000z

/*************************************************/
/*  STRUCTURES & ENUMERATIONS                    */
/*************************************************/

typedef struct temp_ctx_t{

    uint8_t readData[2];
}  TEMP_CTX_T;


/*************************************************/
/*  FUNCTION PROTOTYPES                          */
/*************************************************/

void     sensor_conf        (void);
uint8_t* read_temperature   (void);
void     twi_init           (void);



#endif /* SENSOR_H_ */
```

```
/*-----------------------------------------------------------------------------
 - FILE: sensor.c                                                             -
 -----------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                              -
 -----------------------------------------------------------------------------
 - DESCRIPTION: This file stablishes an interface between application and TWI
   drivers. Contains the information of the temperature sensor, configures it
   and send read request.
 -----------------------------------------------------------------------------*/

#include "sensor.h"


/*****************************************************************************/
/* GLOBAL VARIABLES                                                          */
/*****************************************************************************/

TEMP_CTX_T temp_ctx;

/*****************************************************************************/
/* CONSTANTS                                                                 */
/*****************************************************************************/

const uint8_t conf_data[] = {
   0x60
```

```
};

const uint8_t slave_temp_mem_addr[SLAVE_MEM_ADDR_LENGTH] = {
   0x00
};

const uint8_t slave_conf_mem_addr[SLAVE_MEM_ADDR_LENGTH] = {
   0x01
};


/************************************************************************
* FUNCTION:
*
* sensor_conf
************************************************************************/
/************************************************************************
* DESCRIPTION:
*
* Configures the temperature sensor registers for a resolution of 12 bits
* and sets the pointer to the temperature register.
************************************************************************/
void sensor_conf (void){

   /* configures the TWI configuration packet*/
   twi_package_t packet = {
      .addr[0]      =  slave_conf_mem_addr[0],
      .addr_length  = (uint8_t)SLAVE_MEM_ADDR_LENGTH,
      .chip         = SLAVE_BUS_ADDR,
      .buffer       = (void *)conf_data,
      .length       = DATA_LENGTH
   };

   /* Perform a multi-byte write access */
   while (twi_master_write(TWI_BASE_REG,&packet) != TWI_SUCCESS) {
   }

   /* waits for write completion*/
   delay_ms(5);
}



/************************************************************************
* FUNCTION:
*
* read_temperature
************************************************************************/
/************************************************************************
* PARAMETERS:
*
* OUT:
* @temp_ctx.readData: temperature data read from the temperature sensor.
************************************************************************/
/************************************************************************
* DESCRIPTION:
*
* This function establishes a TWI communication with the temperature
* sensor for read the temperature data.
************************************************************************/
uint8_t* read_temperature (void){

   uint8_t received_data[2] = {0, 0};

   /* configures the TWI read packet*/
   twi_package_t packet_received = {
      .addr[0]      = slave_temp_mem_addr[0],
      .addr_length  = (uint8_t)SLAVE_MEM_ADDR_LENGTH,
```

```
      .chip          = SLAVE_BUS_ADDR,
      .buffer        = received_data,
      .length        = 2
   };

   /* Perform a multi-byte read access*/
   while (twi_master_read(TWI_BASE_REG,&packet_received) != TWI_SUCCESS) {
   }

   temp_ctx.readData[0] = received_data[0];
   temp_ctx.readData[1] = received_data[1];

   return  temp_ctx.readData;

}


/*************************************************************************
* FUNCTION:
*
* twi_init
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* initializes the behavior of the TWI communication.
*************************************************************************/
void twi_init (void){

   /* TWI master initialization options. */
   twi_master_options_t opt = {
      .speed      = TWI_SPEED_HZ,
      .chip  = 0x40,
   };
   opt .baud_reg = TWI_CLOCK_RATE(sysclk_get_cpu_hz(), opt.speed);

   /* Enable the peripheral clock for TWI module */
   sysclk_enable_peripheral_clock(TWI_BASE_REG);

   /* Initialize the TWI master driver. */
   twi_master_init(TWI_BASE_REG,&opt);
}
```

```
/*-------------------------------------------------------------------------
 - FILE: twi.h                                                            -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
 - DESCRIPTION: Header file which contains the definitions, enumerations,
   structures and  function prototypes used in twi.c file.
 -------------------------------------------------------------------------*/

#ifndef _TWI_H_
#define _TWI_H_


#include "compiler.h"
#include "status_codes.h"
#include <conf_twi.h>
```

```
/*************************************************/
/*  DEFINITIONS                                  */
/*************************************************/

/* states of the TWI communication */
#define TWI_IDLE                    1
#define TWI_WRITE_IADDR_WRITE_DATA  2
#define TWI_WRITE_IADDR_READ_DATA   3
#define TWI_WRITE_DATA              4
#define TWI_READ_DATA               5
#define TWI_TRANSAC_SUCCESS         6
#define TWI_TRANSAC_FAIL            7
#define TWI_PROCESS                 8



/* status of the twi transaction */
#define TWI_SUCCESS                 0
#define TWI_STATUS_NO_STATE         1
#define TWI_STATUS_TX_COMPLETE      2
#define TWI_STATUS_RX_COMPLETE      3
#define TWI_STATUS_IO_ERROR        -1
#define TWI_STATUS_PROTOCOL_ERROR  -2

#define TWI_TWSR_STATUS_MASK    (TWSR & 0xF8)
#define TWI_TWSR_PRESCALE0          0x00
#define TWI_PRESCALE_REG            TWI_TWSR_PRESCALE

/** TWI baud rate calculation. Formula described in datasheet. */
#define TWI_CLOCK_RATE(F_CPU, \
TWI_SPEED) (((F_CPU / \
TWI_SPEED) - 16) / (2 * TWI_PRESCALER))


/*************************************************/
/*  STRUCTURES & ENUMERATIONS                    */
/*************************************************/

typedef struct {

   unsigned long speed;
   uint8_t baud_reg;
   char chip;
} twi_master_options_t;

typedef struct {
   uint8_t *rx_buffer;
   uint8_t *tx_buffer;
} slave_data_buffer_t;

typedef struct {
   char chip;
   uint8_t addr[3];
   int addr_length;
   uint8_t *buffer;
   unsigned int length;
} twi_package_t;


/** TWI status codes */
enum {
   TWS_BUSERROR      = 0x00,
   TWS_START         = 0x08,
   TWS_RSTART        = 0x10,
   TWS_MT_SLA_ACK    = 0x18,
   TWS_MT_SLA_NACK   = 0x20,
```

```
   TWS_MT_DATA_ACK    = 0x28,
   TWS_MT_DATA_NACK   = 0x30,
   TWS_M_ARB_LOST     = 0x38,
   TWS_MR_SLA_ACK     = 0x40,
   TWS_MR_SLA_NACK    = 0x48,
   TWS_MR_DATA_ACK    = 0x50,
   TWS_MR_DATA_NACK   = 0x58,


};

/*****************************************************/
/*  INLINE FUNCTIONS                                 */
/*****************************************************/

static inline void twi_interrupt_enable(void)
{ TWCR |= (1 << TWIE); }

static inline void twi_interrupt_disable(void)
{ TWCR &= (~(1 << TWIE)); }

static inline void twi_reset(void)
{ TWCR = ((1 << TWSTO) | (1 << TWINT)); }

static inline uint8_t twi_read_byte(void)
{ return TWDR; }

static inline void twi_write_byte(uint8_t data)
{
   TWDR =  data;
   TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWIE);
}

static inline void twi_send_ack(bool ack)
{
   if (ack) {
      TWCR |= (1 << TWEA);
   } else {
      TWCR &= ~(1 << TWEA);
   }

   TWCR |= ((1 << TWINT) | (1 << TWIE) | (1 << TWEN));
}

static inline void twi_send_stop(void)
{ TWCR = ((1 << TWSTO) | (1 << TWINT) | (1 << TWEN)); }

static inline void twi_send_start(void)
{ TWCR = ((1 << TWSTA) | (1 << TWINT) | (1 << TWEN) | (1 << TWIE)); }



/*****************************************************/
/*  FUNCTION PROTOTYPES                              */
/*****************************************************/

status_code_t twi_master_write (volatile void *twi,
                                const twi_package_t *package);

status_code_t twi_master_read  (volatile void *twi,
                                const twi_package_t *package);

status_code_t twi_master_get_status(void);

status_code_t twi_master_init  (volatile void *twi,
                                twi_master_options_t *opt);

#endif  /* _TWI_H_*/
```

```
/*-------------------------------------------------------------------------
 - FILE: twi.c                                                            -
 -------------------------------------------------------------------------
 - AUTHOR: Jesus Rodriguez Lopez                                          -
 -------------------------------------------------------------------------
 - DESCRIPTION: This file contains the drivers of twi module. Performs
   writting and reading operations as master in the communication.
 -------------------------------------------------------------------------*/

#include "twi.h"

#define MASTER    0


/** Master Transfer Descriptor */
static struct {

   twi_package_t *pkg;
   int addr_count;
   unsigned int data_count;
   uint8_t state;
   status_code_t status;
}  master_transfer;


static volatile bool twi_master_busy = false;
static volatile bool twi_mode = MASTER;


/*************************************************************************
* FUNCTION:
*
* twi_master_bus_reset
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* reset TWI bus.
*************************************************************************/
static void twi_master_bus_reset(void)
{
   master_transfer.state = TWI_IDLE;
   twi_master_busy      = false;
   twi_reset();
}


/*************************************************************************
* FUNCTION:
*
* twi_master_read_last_byte
*************************************************************************/
/*************************************************************************
* PARAMETERS:
*
* IN:
* @data: contains byte that was read.
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* notification that last byte was read from the TWI. Needs to send STOP
* condition.
```

```
*****************************************************************************/
static void twi_master_read_last_byte(uint8_t data)
{
   if (TWI_READ_DATA == master_transfer.state) {
      master_transfer.pkg->buffer[master_transfer.data_count++] = data;
      twi_send_stop();
      master_transfer.state = TWI_IDLE;
      master_transfer.status = STATUS_OK;
      twi_master_busy  = false;
   } else {
      twi_master_bus_reset();
      master_transfer.status = ERR_PROTOCOL;
   }
}


/*****************************************************************************
* FUNCTION:
*
* twi_master_read_done
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @data: contains byte that was read.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* notification that byte was read by the TWI.
*****************************************************************************/
static void twi_master_read_done(uint8_t data)
{
   if (TWI_READ_DATA == master_transfer.state) {
      master_transfer.pkg->buffer[master_transfer.data_count++] = data;
      if (master_transfer.data_count < (master_transfer.pkg->length - 1)) {
         twi_send_ack(true);
      } else {
         twi_send_ack(false);
      }
   } else {
      twi_master_bus_reset();
      master_transfer.status = ERR_PROTOCOL;
   }
}

/*****************************************************************************
* FUNCTION:
*
* twi_master_addr_ack
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* notification that address byte was written to the TWI and need to send
* ACK or NACK.
*****************************************************************************/
static void twi_master_addr_ack(void)
{
   if (TWI_READ_DATA == master_transfer.state) {
      if (master_transfer.data_count == (master_transfer.pkg->length - 1)) {
         twi_send_ack(false);
      } else {
         twi_send_ack(true);
      }
   } else {
      twi_master_bus_reset();
```

```
      master_transfer.status = ERR_PROTOCOL;
   }
}

/*************************************************************************
* FUNCTION:
*
* twi_master_internal_addr_write
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* sending internal device address to twi bus.
*************************************************************************/
static void twi_master_internal_addr_write(void)
{
   uint8_t data;

   data = master_transfer.pkg->addr[master_transfer.addr_count];
   master_transfer.addr_count++;
   twi_write_byte(data);

   if (master_transfer.pkg->addr_length == master_transfer.addr_count) {
      if (TWI_WRITE_IADDR_WRITE_DATA == master_transfer.state) {
         master_transfer.state = TWI_WRITE_DATA;
      } else {
         master_transfer.state = TWI_READ_DATA;
      }
   }
}

/*************************************************************************
* FUNCTION:
*
* twi_master_data_write
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* sending data to twi bus. If last byte then send stop condition.
*************************************************************************/
static void twi_master_data_write(void)
{
   if (master_transfer.data_count < master_transfer.pkg->length) {
      twi_write_byte(master_transfer.pkg-
>buffer[master_transfer.data_count++]);
   } else {
      twi_send_stop();
      master_transfer.state = TWI_IDLE;
      master_transfer.status = STATUS_OK;
      twi_master_busy = false;
   }
}

/*************************************************************************
* FUNCTION:
*
* twi_master_write_done
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* notification that byte was written to the TWI.
*************************************************************************/
static void twi_master_write_done(void)
{
   if (TWI_WRITE_DATA == master_transfer.state) {
      twi_master_data_write();
```

```
   } else if ((TWI_WRITE_IADDR_WRITE_DATA == master_transfer.state) ||
         (TWI_WRITE_IADDR_READ_DATA == master_transfer.state)) {
      twi_master_internal_addr_write();
   } else if (TWI_READ_DATA == master_transfer.state) {
      twi_send_start();
   } else {
      twi_master_bus_reset();
      master_transfer.status = ERR_PROTOCOL;
   }
}


/*****************************************************************************
* FUNCTION:
*
* twi_master_start
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* notification about the start condition was sent.
*****************************************************************************/
static void twi_master_start(void)
{
   uint8_t chip_add;

   if ((TWI_WRITE_IADDR_WRITE_DATA == master_transfer.state) || (TWI_WRITE_DATA
==
         master_transfer.state) || (TWI_WRITE_IADDR_READ_DATA ==
         master_transfer.state)) {
      chip_add = TWI_WRITE_ENABLE(master_transfer.pkg->chip);
      twi_write_byte(chip_add);
   } else if (TWI_READ_DATA == master_transfer.state) {
      chip_add = TWI_READ_ENABLE(master_transfer.pkg->chip);
      twi_write_byte(chip_add);
   } else {
      twi_master_bus_reset();
      master_transfer.status = ERR_PROTOCOL;
   }
}

/*****************************************************************************
* FUNCTION:
*
* twi_master_write
*****************************************************************************/
/*****************************************************************************
* PARAMETERS:
*
* IN:
* @package: package information and data
*
* OUT:
* @status_code_t: result of the requested write operation.
*****************************************************************************/
/*****************************************************************************
* DESCRIPTION:
*
* perform a TWI master write transfer.
*****************************************************************************/
status_code_t twi_master_write(volatile void *twi,const twi_package_t
*package)
{
   /* check the function arguments. */
   if (package == NULL) {
      return ERR_INVALID_ARG;
   }
```

```
   if (twi_master_busy == true) {
      return OPERATION_IN_PROGRESS;
   }

   /* Initiate a transaction when the bus is ready. */
   master_transfer.pkg  = (twi_package_t *)package;
   master_transfer.addr_count  = 0;
   master_transfer.data_count  = 0;
   twi_master_busy = true;

   if (TWI_SLAVE_NO_INTERNAL_ADDRESS == master_transfer.pkg->addr_length) {
      master_transfer.state  = TWI_WRITE_DATA;
   } else {
      master_transfer.state  = TWI_WRITE_IADDR_WRITE_DATA;
   }

   twi_send_start();

   while(twi_master_busy);

   return twi_master_get_status();
}


/*************************************************************************
* FUNCTION:
*
* twi_master_read
*************************************************************************/
/*************************************************************************
* PARAMETERS:
*
* IN:
* @package: package information and data
*
* OUT:
* @status_code_t: result of the requested read operation.
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* reads the series of bytes from the TWI bus.
*************************************************************************/
status_code_t twi_master_read(volatile void *twi,const twi_package_t *package)
{
   /* check the arguments. */
   if (package == NULL ) {
      return ERR_INVALID_ARG;
   }

   if (true == twi_master_busy) {
      return OPERATION_IN_PROGRESS;
   }

   /* Initiate a transaction when the bus is ready. */
   master_transfer.pkg         = (twi_package_t *)package;
   master_transfer.addr_count  = 0;
   master_transfer.data_count  = 0;
   twi_master_busy      = true;

   if (TWI_SLAVE_NO_INTERNAL_ADDRESS == master_transfer.pkg->addr_length) {
      master_transfer.state = TWI_READ_DATA;
   } else {
      master_transfer.state = TWI_WRITE_IADDR_READ_DATA;
   }

   twi_send_start();
```

```
   while(twi_master_busy);

   return twi_master_get_status();
}

/*************************************************************************
* FUNCTION:
*
* twi_master_get_status
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* returns the status of TWI bus
*************************************************************************/
status_code_t twi_master_get_status(void)
{
   return master_transfer.status;
}


/*************************************************************************
* FUNCTION:
*
* twi_master_init
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* this function is a TWI Master initialization.
*************************************************************************/
status_code_t twi_master_init(volatile void *twi, twi_master_options_t *opt)
{
   cpu_irq_disable();

   TWCR = 0x00;
   TWSR = TWI_PRESCALE_REG;
   TWBR = opt->baud_reg;

   twi_interrupt_enable();

   cpu_irq_enable();

   twi_mode = MASTER;

   return STATUS_OK;
}


/*************************************************************************
* FUNCTION:
*
* twi_interrupt_handler
*************************************************************************/
/*************************************************************************
* DESCRIPTION:
*
* TWI interrupt routine which handles the communication with a slave.
* The TWI registers only can be wrote or read in this function and uses
* the TWI status register to know the next steep in the communication.
*************************************************************************/
static void twi_interrupt_handler(void)
{
   uint8_t status;
   status = TWI_TWSR_STATUS_MASK;

   switch (status) {
   case TWS_START:
```

```
   case TWS_RSTART:
      twi_master_start();
      break;

   case TWS_MT_SLA_ACK:
   case TWS_MT_DATA_ACK:
      twi_master_write_done();
      break;

   case TWS_BUSERROR:
   case TWS_MT_SLA_NACK:
   case TWS_MT_DATA_NACK:
   case TWS_MR_SLA_NACK:

      twi_master_bus_reset();
      master_transfer.status = ERR_IO_ERROR;
      break;

   case TWS_MR_SLA_ACK:
      twi_master_addr_ack();
      break;

   case TWS_MR_DATA_ACK:
      twi_master_read_done(twi_read_byte());
      break;

   case TWS_MR_DATA_NACK:
      twi_master_read_last_byte(twi_read_byte());
      break;

   case TWS_M_ARB_LOST:
       master_transfer.state = TWI_IDLE;
      master_transfer.status = ERR_BUSY;
      twi_master_busy = false;
      break;

   default:
         master_transfer.state = TWI_IDLE;
         master_transfer.status = ERR_PROTOCOL;
         twi_master_busy = false;

      break;
   }
}

/**
 * \brief TWI interrupt Vector
 */
/************************************************************************
* FUNCTION:
*
* ISR for TWI_vect
************************************************************************/
/************************************************************************
* DESCRIPTION:
*
* calls the twi interrupt handler function.
************************************************************************/
ISR(TWI_vect)
{
   twi_interrupt_handler();
}
```

15.3. DIAGRAMA DEBLOQUES Y ESQUEMÁTICOS DE LA PLATAFORMA ATMEGA256RFR2 XPLAINED PRO

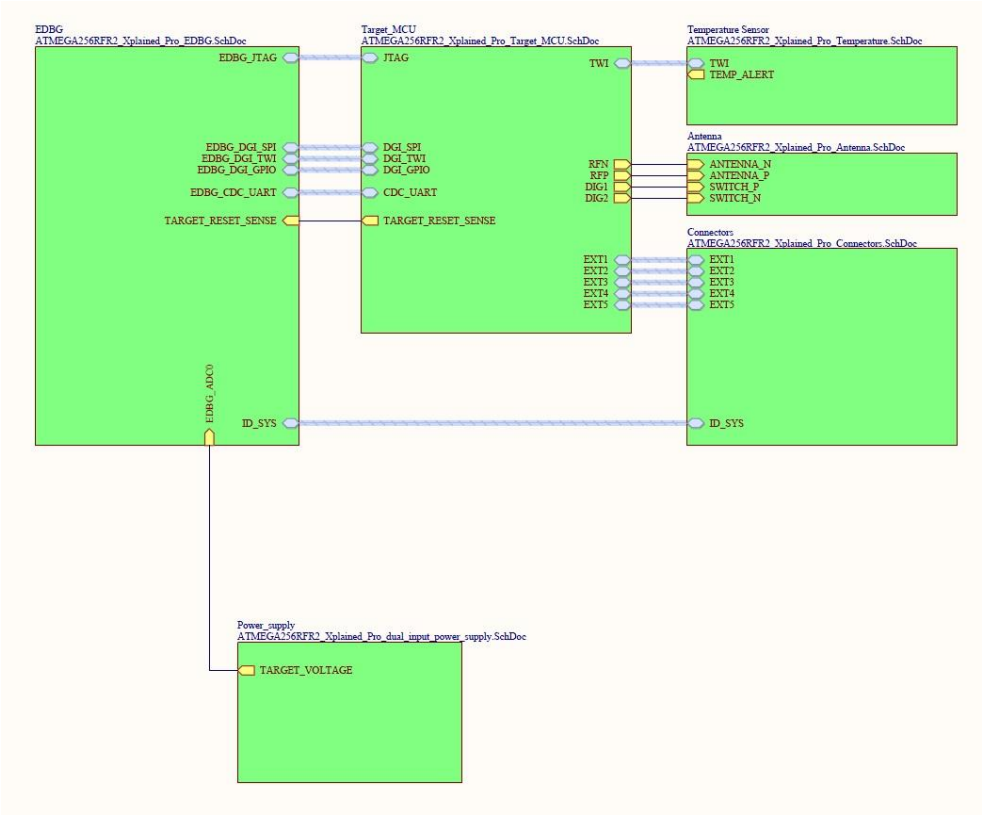

Figura 59. Diagrama de bloques de la plataforma ATMEGA256RFR2 Xplained Pro.

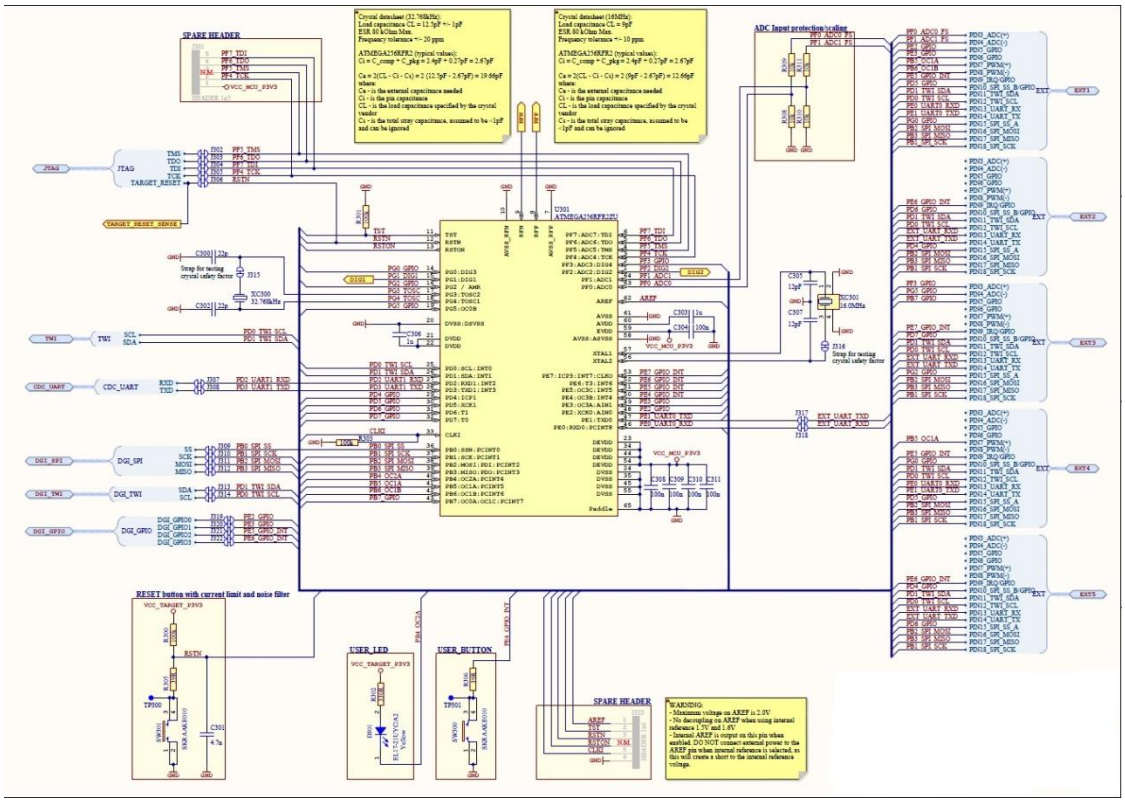

Figura 60. Esquemático del microcontrolador.

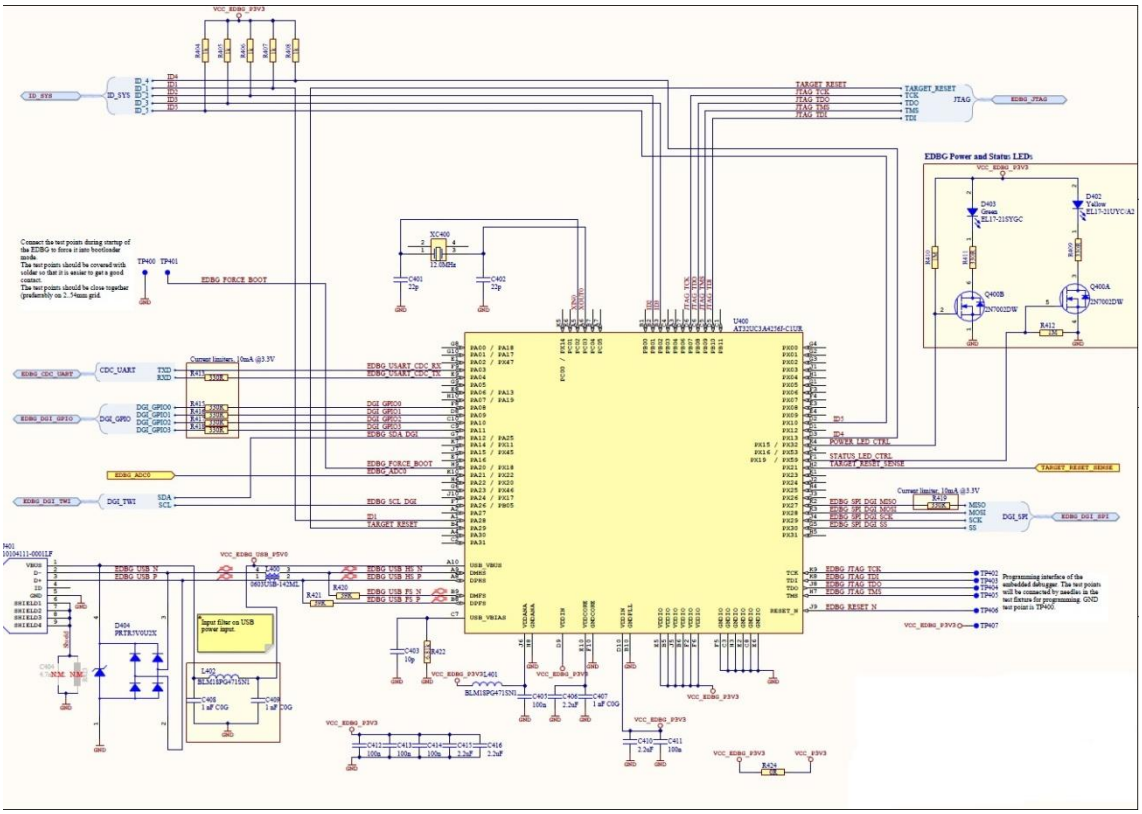

Figura 61. Esquemático del depurador integrado en la plataforma.

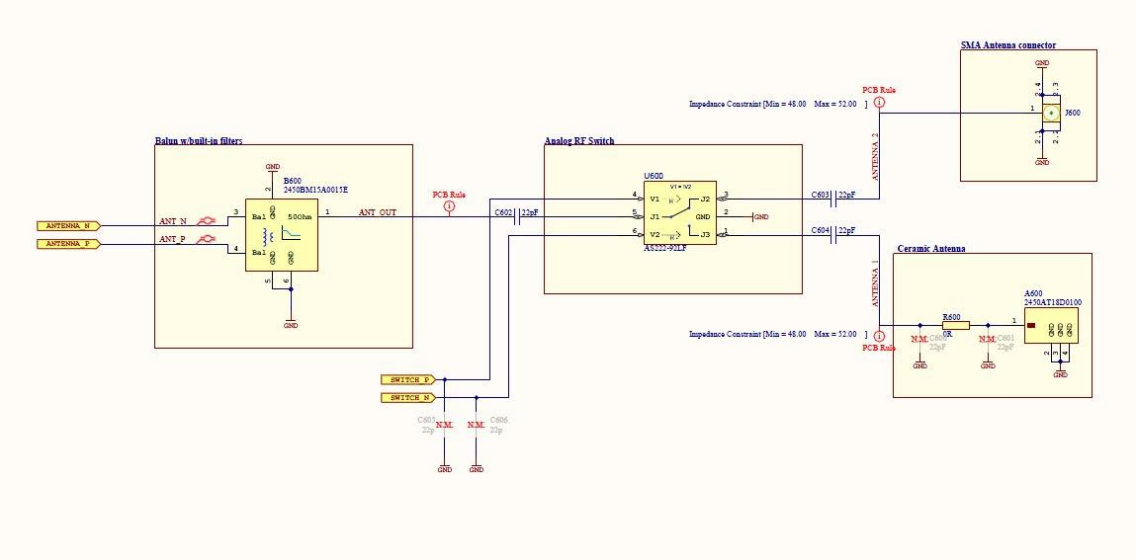

Figura 62. Esquemático de las antenas.

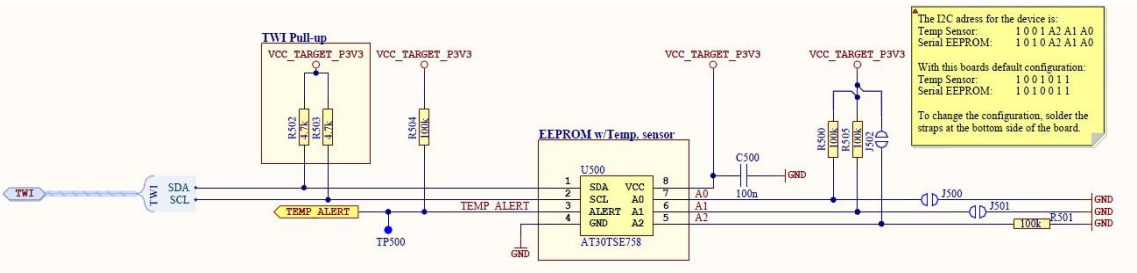

Figura 63. Esquemático del sensor de temperatura.